5-19, 25
4-35
3-23

CONTENTS

KCS–101/201 : Programming for Problem Solving

1
UNIT

Introduction to Programming

PART-1

Introduction to Components of a Computer System : Memory, Processor, I/O Devices, Storage, Operating System, Concept of Assembler, Compiler, Interpreter, Loader and Linker.

CONCEPT OUTLINE : PART-1

- Components of computer system :
  - i. Hardware
  - ii. Operating system
  - iii. Application programs
  - iv. Users
- The memory of a computer holds (stores) program instruction (what to do), data (information), operand (manipulated or operated upon data), and calculations (ALU results).
- An operating system act as an intermediary between the user of a computer and computer hardware.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.1. **Explain the functional units of digital system and their interconnections.**

OR

Discuss various functional components of digital computer.

AKTU 2016-17(Sem-2), Marks 10

OR

What is digital computer ? Draw block diagram of digital computer and explain each components of it.

AKTU 2017-18(Sem-1), Marks 07

Answer

Digital computer is an electronic computer in which the input is discrete rather than continuous, consisting of combinations of numbers, letters and other characters written in an appropriate programming language and represented internally in binary notation.

The main functional units/components of a digital computer are :

1. Central Processing Unit (CPU) :

a. The CPU is the brain of a computer system.

b. This unit takes the data from the input devices and processes it according to the set of instructions called program.

c. The output of processing of the data is directed to the output devices for use in the outside world.

d. CPU has two major parts called ALU and Control Unit.

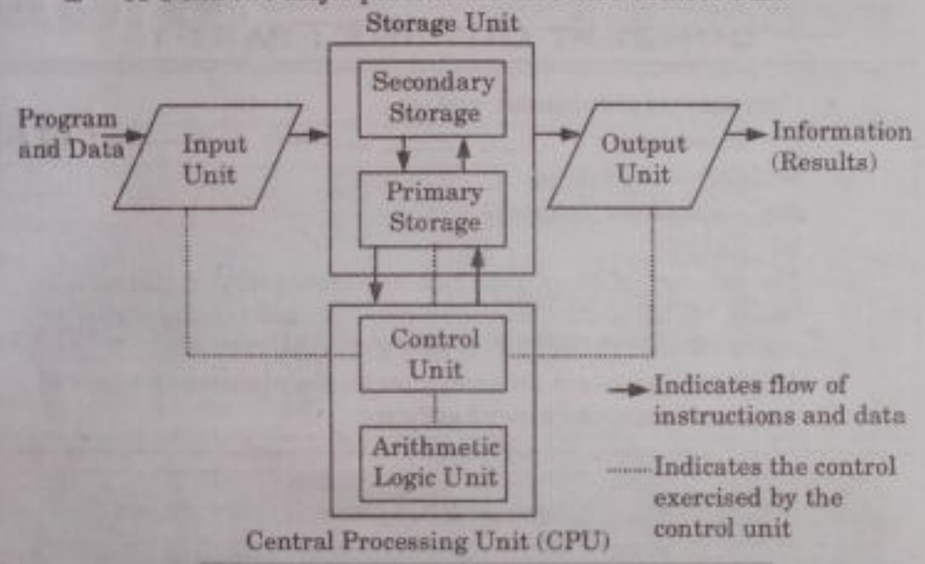


Fig. 1.1.1. Functional unit of digital computer.

i. Arithmetic Logic Unit (ALU) :

1. ALU is responsible for carrying out following operations :
   a. Arithmetic operations on data by adding, subtracting, multiplying and dividing one set with another.
   b. Logical operations by using AND, OR, NOT and exclusive-OR operation which is done by analyzing and evaluating data.
2. ALU of a computer system is the place where actual execution of instructions takes place during processing operation.

ii. Control Unit (CU) :

1. This unit is mainly used for generating the electronic control signals for the synchronization of various operations.
2. All the related functions for program execution such as memory read, memory write, I/O read, I/O write, execution of instruction, are synchronized through the control signal generated by the control unit.
3. It manages and controls all the operations of the computer.

2. Input unit :

An input unit performs following functions :

a. It accepts (or reads) instructions and data from outside world.

b. It converts these instructions and data in computer acceptable form.

c. It supplies the converted instructions and data to computer system for further processing.

3. Output unit :

An output unit performs following functions :

a. It accepts the results produced by a computer, which are in coded form.

b. It converts these coded results to human acceptable (readable) form.

c. It supplies the converted results to outside world.

4. Storage unit :

A storage unit holds (stores) :

a. Data and instructions required for processing (received from input devices).

b. Intermediate results of processing.

c. Results for output, before they are released to an output device.

Que 1.2. What are the components of computer system ?

OR

Describe about the basic components of a computer with a neat block diagram. **AKTU 2014-15(Sem-1), 2016-17(Sem-1); Marks 10**

Answer

Components of computer system are :

1. **Hardware :** Provides basic computing resources (CPU, memory, I/O devices).

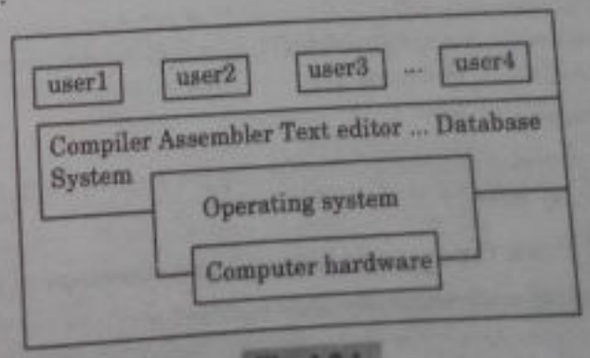


Fig. 1.2.1.

2. **Operating system :** Controls and coordinates the use of the hardware among the various application programs for the various users.
3. **Application programs :** Define the ways in which the system resources are used to solve the computing problems of the users (compilers, database systems, video games, business programs).
4. **Users :** people, machines, other computers.

Que 1.3. What are the basic structural elements of computer system ?

Answer

There are four basic structural elements of computer system which are following :

1. **Processor :**
   a. It controls the operation of the computer and performs its data processing functions.
   b. When there is only one processor, it is often referred to as the Central Processing Unit (CPU).
2. **Main memory :**
   a. It stores data and programs.
   b. This memory is typically volatile *i.e.*, when the computer is shutdown, the contents of the memory are lost.
   c. In contrast, the contents of disk memory are retained even when the computer system is shutdown.
   d. Main memory is also referred to as real memory or primary memory.
3. **I/O modules :**
   a. They move data between the computer and its external environment.
   b. The external environment consists of a variety of devices, including secondary memory devices (For example : disks), communications equipment, and terminals.
4. **System bus :**
   a. It provides communication among processors, main memory, and I/O modules.

Que 1.4. Draw the memory hierarchical structure of a computer system. Explain each memory unit in brief.

AKTU 2014-15(Sem-2), Marks 05

Answer

1. The memory hierarchy system consists of all storage devices employed in a computer system from the slow but high capacity auxiliary memory to a relatively faster main memory, to as even smaller and faster cache memory accessible to the high speed processing logic.
2. Fig. 1.4.1 shows the typical memory hierarchy :

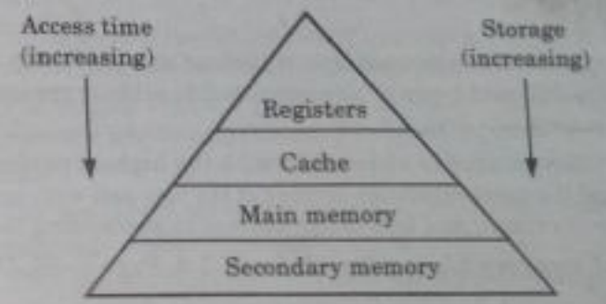


Fig. 1.4.1. Memory hierarchy.

Memory hierarchy is layered into these layers :

1. **Registers :**
   a. Registers are special memory units which are used to handle the information between various units of the computer and to speed up the rate of information transfer.
   b. These registers are used to hold information on temporary basis.
2. **Cache :**
   a. Cache is used by the CPU for memory which is being accessed over and over again.
   b. Instead of pulling it every time from the main memory, it is put in cache for fast access.
3. **Main memory :**
   a. The main memory of the computer is also known as RAM (Random Access Memory).
   b. It is constructed from integrated circuits and needs to have electrical power in order to maintain its information.
   c. When power is lost, the information is lost too. It can be directly accessed by the CPU.
4. **Secondary/Auxiliary memory :**
   a. Secondary memory is computer memory that is non-volatile and persistent in nature and is not directly accessed by a computer/ processor.

b. It allows a user to store data that may be instantly and easily retrieved, transported and used by applications and services.

c. The most common forms of secondary/auxiliary memory are flash memory, optical disks, magnetic disks and magnetic tape.

Que 1.5. Why is memory system of a computer organized as a hierarchy ? Discuss the basic elements of a memory hierarchy.

Answer

1. Memory system of a computer is organized as a hierarchy to optimize the use of different types of memories and to achieve greater efficiency and economy.
2. Memory is organized in a hierarchy with the highest performance and in general the most expensive devices at the top, and with progressively lower performance and less costly devices in succeeding layers.

Elements of memory hierarchy : Refer Q. 1.4, Page 1–5E, Unit-1.

Que 1.6. Describe I/O devices.

Answer

Input device : Input device can be connected to the computer system using cables. The most commonly used input devices are :

1. **Keyboard :** A standard keyboard includes alphanumeric keys, function keys, modifier keys, cursor movement keys, spacebar, escape key, numeric keypad, and some special keys, such as Page Up, Page Down, Home, Insert, Delete and End.
2. **Mouse :**
   a. The mouse is also known as a pointing device because it helps to change the position of the pointer or cursor on the screen.
   b. The mouse allows us to select elements on the screen, such as tools, icons, and buttons, by pointing and clicking them.
   c. We can also use a mouse to draw and paint on the screen of the computer system.
3. **Scanner :**
   a. A scanner is an input device that converts documents and images as the digitized images understandable by the computer system.
   b. The digitized images can be produced as black and white images, gray images, or coloured images.

Output devices : The data, processed by the CPU, is made available to the end user by the output devices. The most commonly used output devices are :

1. **Monitor :**
   a. A monitor is the most commonly used output device that produces visual displays generated by the computer.
   b. The monitor, also known as a screen connected using cables, is connected to the video card placed on the expansion slot of the motherboard. The display device is used for visual presentation of textual and graphical information.
2. **Printer :**
   a. The printer transfer the text displayed on the screen, onto paper sheets that can be used by the end user.
   b. The various types of printer used are dot matrix printers, inkjet printers, and laser printers.
3. **Speaker :**
   a. The speaker is an electromechanical transducer that converts an electrical signal into sound.
   b. They are attached to a computer as output devices, to provide audio output, such as warning sounds and internet audios.
4. **Plotter :**
   a. The plotter is connected to a computer to print large documents, such as engineering or constructional drawings.
   b. Plotters use multiple link pens or inkjets with colour cartridges for printing.
   c. Plotters are classified on the basis of their performance, as follows :
      i. Drum plotter ii. Flat-bed plotter
      iii. Inkjet plotter iv. Electrostatic plotter

Que 1.7. Describe processor.

Answer

1. A processor is an integrated electronic circuit that performs the calculations that run a computer.
2. A processor performs arithmetical, logical input/outputs and other basic instruction that are passed from an operating system.
3. The CPU consists of :
   a. CU which stores the instruction set, and specifies the operations to be performed by the computer.
   b. ALU which performs arithmetical or logical operations on the data received.
4. The CPU registers store the data to be processed by the CPU and the processed data also.
5. Apart from CU and ALU, CPU seeks help from the following hardware devices to process the data :

i. **Motherboard :** It refers to a device used for connecting the CPU with the input and output devices. The components on the motherboard are connected to all parts of a computer and are kept insulated from each other. Some of the components of a motherboard are :

a. **Buses :** Electrical pathways that transfer data and instructions among different parts of the computer.

b. **System clock :** It is a clock used for synchronizing the activities performed by the computer.

c. **Microprocessor :** CPU component that performs the processing and controls the activities performed by the different parts of the computer.

d. **ROM :** Chip that contains the permanent memory of the computer that stores information, which cannot be modified by the end user.

ii. **Random Access Memory (RAM) :** It refers to primary memory of a computer that stores information and programs, until the computer is used. RAM is available as a chip that can be connected to the RAM slots in the motherboard.

iii. **Video Card/Sound Card :** The video card is an interface between the monitor and the CPU. Video cards also include their own RAM and microprocessors that are used for speeding up the processing and display of a graphic.

A sound card is a circuit board placed on the motherboard and is used to enhance the sound capabilities of a computer.

Que 1.8. **What are the objectives and major functions of an operating system ?**

OR

Describe the functionalities of an operating system.

AKTU 2014-15(Sem-1), Marks 05

OR

What is an operating system ? Describe the functionalities of operating system.

AKTU 2014-15(Sem-2), Marks 05

OR

What is operating system ? Explain various types of functions performed by an operating system.

AKTU 2017-18(Sem-1), Marks 07

Answer

Operating system :

1. An operating system is a software that manages the computer hardware.

2. An operating system acts as an intermediary between the user of a computer and computer hardware.
3. The hardware must provide appropriate mechanisms to ensure the correct operation of the computer system and to prevent user programs from interfering with the proper operation of the system.
4. The purpose of an operating system is to provide an environment in which a user can execute programs in a convenient and efficient manner.

Objectives of OS :

1. **Convenience :** An OS makes a computer more convenient to use.
2. **Efficiency :** An OS allows the computer system resources to be used in an efficient manner.
3. **Ability to evolve :** An OS should be constructed in such a way as to permit the effective development, testing, and introduction of new system functions without interfering with service.

Functions of an operating system are :

1. **Memory management :**
   a. Memory management refers to management of primary memory or main memory.
   b. An operating system does the following activities for memory management :
      i. Keeps tracks of primary memory, *i.e.*, what part of it are in use by whom, what part are not in use.
      ii. In multi-programming, the OS decides which process will get memory when and how much.
      iii. Allocates the memory when a process requests it to do so.
      iv. De-allocates the memory when a process no longer needs it or has been terminated.
2. **Processor management :**
   a. In multi-programming environment, the OS decides which process gets the processor when and for how much time. This function is called process scheduling.
   b. An operating system does the following activities for processor management :
      i. Keeps tracks of processor and status of process. The program responsible for this task is known as traffic controller.
      ii. Allocates the processor (CPU) to a process.
      iii. De-allocates processor when a process is no longer required.
3. **Device management :**
   a. An operating system manages device communication via their respective drivers. It does the following activities for device management :

i. Keeps tracks of all devices. Program responsible for this task is known as the I/O controller.

ii. Decides which process gets the device when and for how much time.

iii. Allocates the device in the efficient way.

iv. De-allocates devices.

4. **File management :**

a. A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions.

b. An operating system does the following activities for file management :

i. Keeps track of information, location, uses, status etc. The collective facilities are often known as file system.

ii. Decides who gets the resources.

iii. Allocates the resources.

iv. De-allocates the resources.

Que 1.9. **Explain the concept of assembler and cross assembler.**

Answer

Assembler :

1. A program which translates an assembly language program into a machine language program is called an assembler.
2. Assembler uses mnemonic symbols that can be easily translated into machine codes.

Cross assembler :

1. A cross assembler is an assembler that runs on a computer other than that for which it produces machine codes.
2. In such a situation, a faster and powerful computer can be used for program development. The programs so developed are to be run on smaller computers. For such program development, a cross assembler is required.
3. These are further divided as follows :

a. **One-pass assembler :** These are equipped to assign the memory addresses to the variables and translate the instruction simultaneously in the first pass itself.

b. **Two-pass assembler :** These read the program statements twice. In the first pass, it reads all the variables and assigns them memory addresses. In the second pass it reads the instructions manipulating the variables and translates them to manipulate the memory addresses.

Que 1.10. **Describe compiler, interpreter, assembler. Write the names of compiler that are used in C programming.**

AKTU 2014-15(Sem-2), Marks 05

Answer

1. **Compiler :**

a. A program which translates a high-level language into a machine language is called a compiler.

b. A compiler checks all kinds of limits, ranges, errors etc. But its program execution time is more, and occupies a larger part of the memory.

c. It has slow speed and low efficiency in memory utilization.

d. A compiler goes through the entire high-level language program once or twice and then translates the entire program into machine codes.

2. **Interpreter :**

a. An interpreter is a program which translates statements of a high-level language program into machine codes.

b. It translates one statement of the program at a time.

c. It reads one statement of a high-level language program translates it into machine code and executes it. Then it reads the next statement of the program again translates and executes it. In this way it proceeds further till all the statements of the program are translated and executed.

3. **Assembler :** Refer Q. 1.9, Page 1–11E, Unit-1.

Name of compiler used in C programming :

i. Interactive C

ii. Lattice C

iii. GCC C

iv. Micro C compiler

v. Hippo-C

vi. Digital mars

Que 1.11. **Describe about linker and loader.**

OR

Explain loading and linking of a program in detail.

AKTU 2015-16(Sem-2), Marks 7.5

Answer

Linker :

a. A linker is a program that links (combines) smaller programs to form a single program.

b. Linking is a process of gathering or accumulating all the other program files and functions together, that are necessary for the program execution.

c. For example, if the programmer uses the sqrt or pow function in the program, then the programmer have to link the object code of this function to the main function, which is included in the math.h library files.

d. While developing a program, subroutines are frequently used. The subroutines are stored in a library file.

e. The linker also links subroutines with the main program. The linker links machine code of the programs.

f. Therefore, it accepts user's program after editor has edited the program and compiler has produced machine codes of the program.

Loader :

a. The loader is a program that loads machine codes of a program into the system memory.

b. An executable file with extension exe is created, after linking the executable object code.

c. Now, after linking the process, loader loads the program from the disk to the primary memory, which is known as loading process.

d. It accepts program either in absolute or relocatable format.

e. If a program is a absolute format (*i.e.*, the actual addresses of the instructions and data are supplied by the programmer), the loader simply loads the program into the system memory.

f. If a program is in relocatable format, the locater assigns specific addresses to each instruction and data before the loader loads the program into memory.

Que 1.12. **Discuss the various functionalities of compiler, linker and loader.**

AKTU 2014-15(Sem-1), Marks 05

Answer

Various functionalities of :

1. **Compiler :**
   a. Compiler is a translator program that converts high level language programs into machine language.
   b. It scans all the lines of source program and lists all the syntax errors at a time.

   c. The compiler produces a printed listing of the source and object programs.
2. **Linker :**
   a. It is a program that links separately compiled modules into one program.
   b. It also combines the functions in the standard C library with the code that we wrote.
   c. The output of the linker is an executable program.
3. **Loader :**
   a. Before a machine language program can be executed, it must be stored in the memory of the computer. The program itself may be stored in a disk.
   b. This program is to be read into memory by another program called a loader, which is already stored in the memory.
   c. The loader is usually stored in a ROM and is automatically moved to the main memory by the hardware of the computer when it is switched on.

Que 1.13. **Differentiate between linker and loader.**

AKTU 2015-16(Sem-1), Marks 2.5

Answer

S. No.	Linker	Loader
1.	A linker is a program that links separately compiled modules into one program.	The loader is a program that loads machine codes of a program into the system memory.
2.	Linker accepts users program after editor has edited the program and compiler has produced machine codes of the program.	Loader accepts programs either in absolute or relocatable format.
3.	The output of the linker is an executable program.	The output of the loader is an object program.
4.	It is easier to develop, test and debug smaller programs.	It is difficult to test and debug the longer programs.

PART-2

Idea of Algorithm : Representation of Algorithm, Flowchart, Pseudocode with Example, From Algorithms to Programs, Sourcecode.

CONCEPT OUTLINE : PART-2

- An algorithm is a set of instructions which describes the steps to be followed to get the solution of a problem.
- A flowchart is diagrammatic representation of the procedure for solving problem.
- Pseudocode is an imitation of actual computer instructions.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.14. **What do you mean by algorithm ? Explain the properties of algorithm.** **AKTU 2014-15(Sem-2), Marks 05**

OR

Describe algorithm. What are the notations used to write an algorithm ?

Answer

1. An algorithm is a set of instructions which describes the steps to be followed to get the solution of a problem.
2. It can be finite step-by-step procedure to achieve a required result.

The notations used to write an algorithm are :

1. **Name of the algorithm :** It specifies the problem to be solved.
2. **Step number :** Identification tag of an instruction and it is an unsigned positive integer.
3. **Explanatory comment :** It follows the step number and describes the operation. It should be written within a pair of square brackets.
4. **Termination :** It specifies the end of the algorithm. It is generally a stop statement and the last instruction in the algorithm.

Properties of an algorithm :

a. **Finiteness :** Algorithm must complete after number of instructions have been executed.
b. **Absence of ambiguity :** Each step must be clearly defined, having only one interpretation.
c. **Definition of sequence :** Each step must have a unique defined preceding and succeeding step.
d. **Input/output :** Number and types of required inputs and output must be specified.
e. **Feasibility :** It must be possible to perform each instruction.

Que 1.15. **What are the characteristics of an algorithm ? Write an algorithm to find the sum of all the numbers divisible by 3 between 11 and 50.** **AKTU 2013-14(Sem-2), Marks 05** **AKTU 2015-16(Sem-1), Marks 10**

Answer

Characteristics of algorithm are :

1. **Input and output :** These characteristics require that an algorithm produces one or more outputs and have zero or more inputs that are externally supplied.
2. **Definiteness :** Each operation must be perfectly clear and unambiguous.
3. **Effectiveness :** This requires that each operation should be effective, *i.e.*, each step can be done by a person using pencil and paper in a finite amount of time.
4. **Termination :** This characteristic requires that an algorithm must terminate after a finite number of operations.

Algorithm :

Step 1 : Initialize sum = 0, a = 11
Step 2 : Repeat step 3, 4, 5, while (a = 50)
Step 3 : If (a% 3 = = 0)
Step 4 : sum = sum + a
Step 5 : Increment a
Step 6 : Print the sum
Step 7 : Stop

Que 1.16. **What is flowchart ? Discuss advantages and disadvantages of flowchart. Draw a flowchart to find the factorial of the given number.**

Answer

Flowchart :

1. A flowchart is diagrammatic representation of the procedure for solving the problem.
2. It is a pictorial representation that a programmer uses for planning the procedure for solution of problem.
3. Once developed and thoroughly checked, the flowchart provides an excellence guide for writing a program.
4. A flowchart illustrates how the program will create the desired output.

Advantages of flowchart :

1. The graphical represe,ntation of algorithm in the flowchart clearly shows how control structure operates.

2. "START" and "STOP" symbols are clearly indicated.
3. Easy to understand.

Disadvantages of flowchart :

1. Complex flowcharts sometimes lead to confusion.
2. Connectivity issues in case of large programs.

Flowchart to find factorial of a given number :

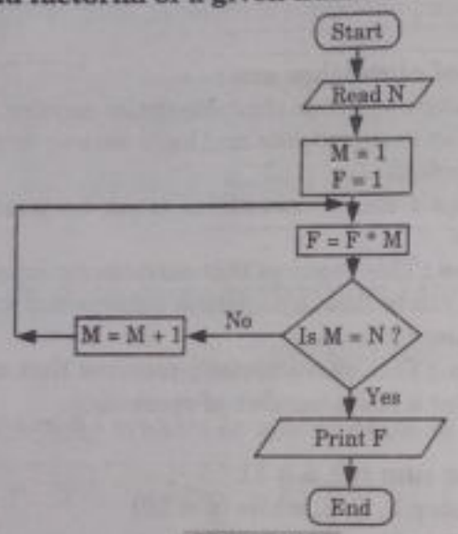


Fig. 1.16.1.

Que 1.17. **What is an algorithm ? Give the characteristics of an algorithm. Write an algorithm for printing table of a given positive integer also draw the flow chart for the same.**

AKTU 2016-17(Sem-2), Marks 07

Answer

Algorithm and its characteristics : Refer Q. 1.14, Page 1–15E and Q. 1.15, Page 1–16E; Unit-1.

Algorithm for printing table of given integer :

Step 1 : Start

Step 2 : Input the number "Num"

Step 3 : Initialize I = 1

Step 4 : Check (I > 10) if yes then Stop else go to step 5

Step 5 : PROD = Num × I

Step 6 : Write PROD

Step 7 : Increment I i.e., I = I + 1

Step 8 : Go to step 4.

Flow chart for printing table of given integer :

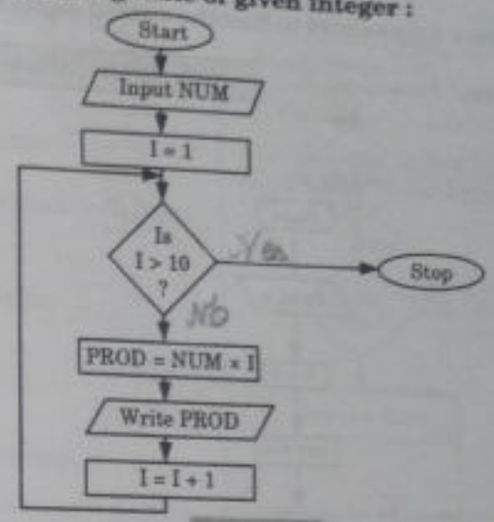


Fig. 1.17.1.

Que 1.18. **Define algorithm and make a flowchart to find prime numbers between 101 and 999.**

AKTU 2013-14(Sem-1), Marks 05

Answer

Algorithm : Refer Q. 1.14, Page 1–15E, Unit-1.

Flowchart for finding prime numbers between 101 and 999 :

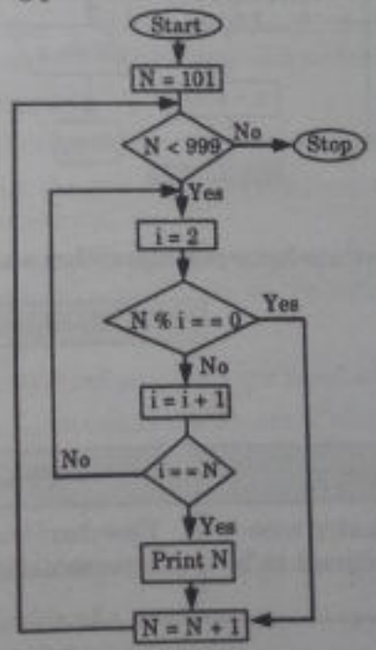


Fig. 1.18.1.

Que 1.19. **Draw a flowchart to find the sum of digits and reverse of a given number.** **AKTU 2014-15(Sem-1), Marks 05**

Answer

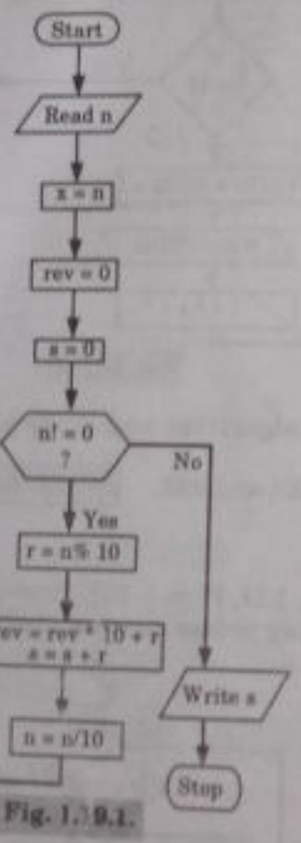


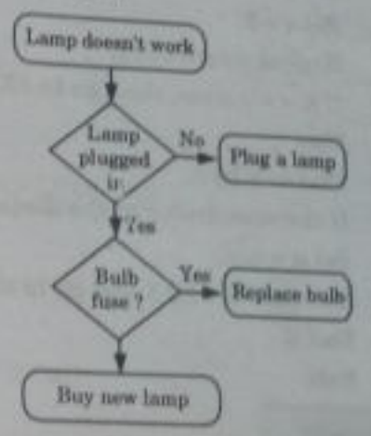
Fig. 1.19.1.

Que 1.20. **Differentiate between algorithm and flowchart.**

AKTU 2015-16(Sem-1), Marks 2.5

Answer

S. No.	Algorithm	Flowchart
1.	Algorithm is a step wise analysis of the work to be done.	Flowchart is a pictorial representation of an algorithm.

2.	**Example :**
	Step 1 : Check lamp plugged in :
	begin
	if(no)
	plug in lamp;
	end;
	Step 2 : Check bulb fuse:
	begin
	if (yes)
	change bulb;
	end;
	Step 3 :
	begin
	else if
	buy new lamp;
	end if;
		Example :
		 

Que 1.21. **Explain pseudocode. Give its advantages and disadvantages.**

Answer

1. Psuedocode is a detailed yet readable description of what a computer program or algorithm must do, expressed in a formally styled natural language rather than in programming language.
2. It is used for creating an outline or a rough draft of a program. It summarizes the program's flow but excludes underlying details.

Advantages of pseudocode :

1. Converting a pseudocode to a programming language is much easier.
2. It is easier to modify the pseudocode of program logic when program modifications are necessary.
3. Pseudocode is easier to write because it has only a few rules to follow, allowing a programmer to concentrate on the logic of the program.

Disadvantage of pseudocode :

1. A graphic representation of program logic is not available.
2. Communication problem occurs due to lack of standardization.
3. For a beginner, it is more difficult to follow the logic of pseudocode or write pseudocode as compared to flowchart.

Que 1.22. **Implement a C program from the given algorithm.**

//To find the prime factor of a number

1. **Read a number, say *n*.**

2. If $n < 1$, the go to step 12.
3. Set $x = 2$.
4. Repeat step 5 to 11.
5. If $n <= x$ num, then go to 12.
6. Else
7. Divide n by x.
8. If the remainder of the division is 0, print x.
9. Set $n = n/x$.
10. Increment x by 1 and go to step 5.
11. End if
12. Exit

Answer

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n, x;
    clrscr();
    printf("Enter a number to find its prime factors:");
    scanf("%d",&n);
    if(n < = 1)
    {
        printf("Enter a value greater than 1.");
        getch();
        exit(0);
    }
    x = 2;
    do
    {
        if(n%x == 0)
        {
            printf("%d\t",x);
            n/=x;
        }
        else
            x++;
    }
    while (x<=n);
    getch();
}
```

Que 1.22. Implement a C program from the given algorithm.

//To find the square root of a number.
1. Read a number, say s.
2. If $s < 0$, then go to step 16.
3. Else if $s = 0$
4. Print the value of sq as 0.
5. Else
6. Set $n = 1$.
7. While (!($s >= n * n$ && $s < (n + 1)*(n + 1)$)
8. Do increment n by 1
9. End while
10. $d = s - (n * n)$
11. P = (double)$d/(2*n)$
12. a = (double)$n + p$
13. root = (double) $a - ((p * p)/(2 * a))$;
14. Print the value of root
15. Endif
16. Exit

Answer

```
#include<stdio.h>
int main()
{
    int s, n;
    double d, p, a, root;
    clrscr();
    printf("Enter a number :");
    scanf("%d", &s);
    if(s<0)
        printf("Enter a positive integer value.");
    else if(s == 0)
        printf("Square root of 0 is 0");
    else
    {
        n = 1;
        while(!(s>=n*n && s<(n+1)*(n+1)))
        {
            n++;
        }
        d=s-(n*n);
        p=(double)d/(2*n);
        d=(double)n+p;
        root=(double)a-((p*p)/(2*a));
        printf("\nSquare root of %d is %.3f",s,root);
    }
    getch();
}
```

PART-3

Programming Basics : Structure of C Program, Writing and Executing the First C Program, Syntax and Logical Errors in Compilation, Object and Executable Code, Components of C Language, Standard I/O in C, Fundamental Datatypes, Variables and Memory Locations, Storage Classes.

CONCEPT OUTLINE : PART-3

- Components of C language :
  - i. Character set
  - ii. Datatypes
  - iii. Constants
  - iv. Variables
  - v. Keywords
- Different types of storage class :
  - i. Automatic storage class
  - ii. Register storage class
  - iii. Static storage class
  - iv. External storage class

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.24. Explain the structure of C program.

Answer

1. Each program begins with preprocessor directives that serve to provide information about functions from standard libraries and definitions of necessary program constants. Examples of such directives are #include and #define.
2. Every C program consists of one or more functions. Function is a self contained block of program which performs some kind of task.
3. Every C program begins with main() function. Every function has a return value.
4. For example, consider the following program which gives summation of x and y as z.

```
#include<stdio.h>                    /* Preprocessor Directive */
```



```
int sum(int,int);                  /* function prototype for sum() */
int a, b, c;                       /* Global Variables */
float d;                           /* Global Variable */
int main()                         /* main() function with return type */
{
int x, y;
int z;                             /* Local variables to main() */
char ch;
z = sum(x,y);
.....                              /* function call */
.....
}
int sum(int x1,int y1)             /* Definition of function sum() */
{
.....
.....
return( );                         /* return value of function sum */
}
```

Que 1.25. How to write and execute a C program ?

Answer

1. C program statements are written in lowercase letters. Uppercase letters are used only for symbolic constants.
2. Braces, group program statements together and mark the beginning and the end of functions. A proper indentation of braces and statements would make a program easier to read and debug.
3. Following program shows how the braces are aligned and the statement is indented :

```
main ()
{
        printf ("hello C");
}
```

4. This program may be written in one line like :

```
main () {printf("Hello C");};
```

However, this cycle makes the program more difficult to understand and should not be used.

5. The generous use of comments inside a program cannot be over emphasized. Judiciously inserted comments not only increase the readability but also help to understand the program logic. This is very important for debugging and testing the program.

Executing a C program :

Executing a program written in C involves a series of step. These are :

1. Creating the program;
2. Compiling the program;
3. Linking the program with functions that are needed from the C library; and
4. Executing the program.

Fig. 1.25.1 illustrates the process of creating, compiling and executing a C program.

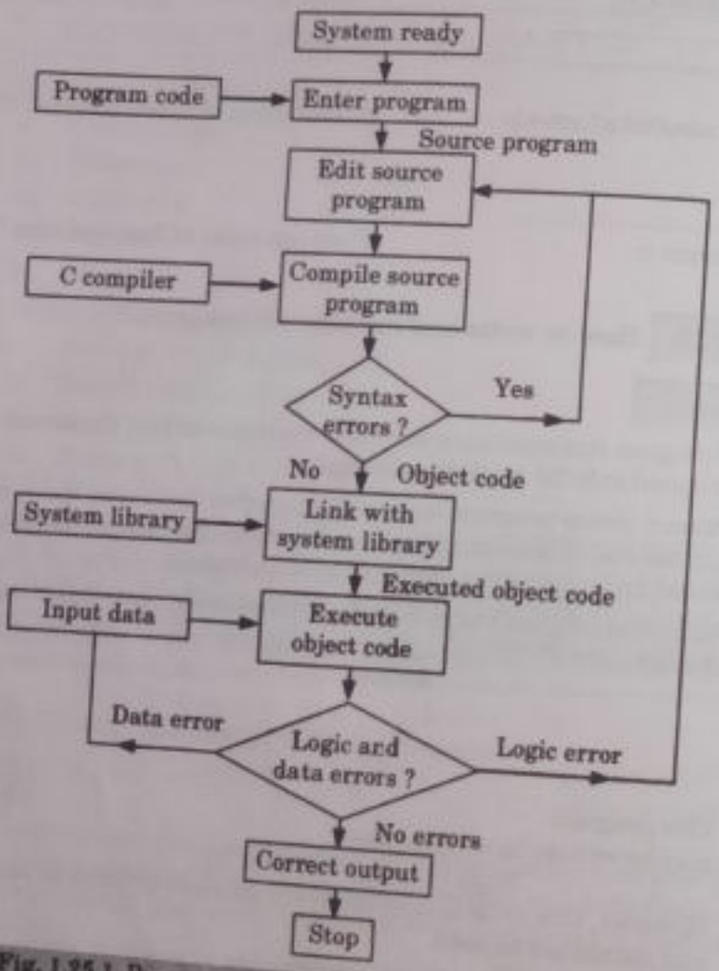


Fig. 1.25.1. Process of compiling and running a C program.

Que 1.26. Describe syntax and logical error which occur during compilation in C with example.

Answer

Syntax errors :

1. Errors that occur when we violate the rules of writing C syntax are known as syntax errors. This compiler error indicates something that must be fixed before the code can be compiled.
2. All these errors are detected by compiler and thus are known as compile-time errors.
3. Most frequent syntax errors are :
   a. Missing Parenthesis (})
   b. Printing the value of variable without declaring it.
   c. Missing semicolon.

For example :

```
// C program to illustrate syntax error
#include<stdio.h>
void main()
{
    int x = 10;
    int y = 15;
    printf("%d", (x, y)) // semicolon missed
}
```

Error :

```
error : expected ';' before '}' token
```

Logical errors :

1. On compilation and execution of a program, desired output is not obtained when certain input values are given. These types of errors which provide incorrect output but appear to be error free are called logical errors.
2. These errors solely depend on the logical thinking of the programmer and are easy to detect if we follow the line of execution and determine why the program takes that path of execution.
3. These are also known as runtime errors.

For example :

```
// C program to illustrate logical error
int main()
{
    int i = 0;
    // logical error : a semicolon after loop
    for (i = 0; i < 3; i++);
    {
        printf("loop ");
        continue;
    }
getchar();
```

```
return 0;
}
```

No output

Que 1.27. Write a short note on object code and executable code.

Answer

Object code :

1. The source program instructions are translated into a form that is suitable for execution by the computer.
2. The translation is done after examining each instruction for its correctness.
3. If everything is alright, the compilation proceeds silently and the translated program is stored on another file with the name a.0.
4. This program is known as object code.

Executable code :

1. If any mistakes in the syntax and semantics of the language are discovered, they are listed out and the compilation process ends right there.
2. The errors should be corrected in the source program with the help of the editor and the compilation is done again.
3. The compiled and linked program is called the executable object code and is stored automatically in another file named a.out.

Que 1.28. Differentiate between logical error and runtime error.

AKTU 2015-16(Sem-1), Marks 2.5

Answer

S. No.	Logical errors	Runtime errors
1.	These errors are related to the logic of the program execution. Failure to consider a particular condition, taking a wrong path, and incorrect order of evaluation of statements belong to this category.	Errors such as mismatch of data type or referencing an out of range array element go undetected by the compiler. A program with such mistakes will run, but produce erroneous results at the time of execution.

Que 1.29. What are the components of C language ?

Answer

Following are the components of C language :

1. **Character set :**

   Any alphabet, digit, or special symbol used to represent information is denoted by a character. The character in C are grouped into four categories :

   a. Letters : A - Z or a - z
   b. Digits : 0, 1 ------ 9
   c. Special symbols : ~ ! @ # % ^ & * ( ) _ - + | \ { } [ ] : ; " ' < > , . ? /
   d. White spaces : blank space, horizontal tab, carriage return, new line, and form feed.

2. **Data type :** C language is designed to handle five primary data types namely character, integer, float, double and void, and several secondary data type like array, pointer, structure, union, and enumeration.
3. **Constants :** A constant is a fixed value entity that does not change. It can be stored at a location in the memory of the computer and can be referenced through that memory address.
4. **Variables :** A variables is an entity whose value can change during program execution. A variable can be thought of as symbolic representation of address of the memory whose values can change.
5. **Keywords :** Keywords are those words which have been assigned specific meaning in the context of C language programs. Keywords should not be used as variable names to avoid problems or compile time errors.

Que 1.30. Describe the standard I/O function in C.

Answer

1. C programming provides a set of built-in functions to read the given input and feed it to the program as per requirement.
2. C programming provides a set of built-in functions to output the data on the computer screen as well as to save it in text or binary files.

Different I/O functions are :

1. **The getchar() and putchar() functions :**
   a. The int getchar(void) function reads the next available character from the screen and returns it as an integer. This function reads only single character at a time.
   b. The int putchar(int c) function puts the passed character on the screen and returns the same character. This function puts only single character at a time.

2. **The gets() and puts() functions :**
   a. The char *gets(char *s) function reads a line from standard input stream stdin into the buffer pointed to by *s* until either a terminating newline or EOF (End of File).
   b. The int puts(const char *s) function writes the string *s* and a trailing newline to standard output stream stdout.
3. **The scanf() and printf() functions :**
   a. The int scanf(const char *format, ...) function reads the input from the standard input stream stdin and scans that input according to the format provided.
   b. The int printf(const char *format, ...) function writes the output to the standard output stream stdout and produces the output according to the format provided.

Que 1.31. **Define data types in C. Discuss primitive data types in terms of memory size, format specifier and range.**

AKTU 2015-16(Sem-1), Marks 05

AKTU 2016-17(Sem-1), Marks 05

AKTU 2017-18(Sem-1), Marks 07

Answer

1. C data types are defined as the data storage format that a variable can store a data to perform a specific operation.
2. Data types are used to define a variable before to use in a program.

Primitive data types in C are :

1. **Integer type :** Integers are used to store whole numbers.

Size and range of integer type on 16-bit machine :

Type	Size(bytes)	Range	Format specifier
int or signed int	2	-32,768 to 32,767	%d
unsigned int	2	0 to 65,535	%u
short int or signed short int	1	– 128 to 127	%hd
unsigned short int	1	0 to 255	%hu
long int or signed long int	4	– 2,147,483,648 to 2,147,483,647	%ld
unsigned long int	4	0 to 4,294,967,295	%lu

2. **Floating point type :** Floating types are used to store real numbers.

Size and range of integer type on 16-bit machine :

Type	Size(bytes)	Range	Format specifier
Float	4	3.4E – 38 to 3.4E+38	%f
double	8	1.7E – 308 to 1.7E+308	%lf
long double	10	3.4E – 4932 to 1.1E+4932	%Lf

3. **Character type :** Character types are used to store characters value.

Size and range of integer type on 16-bit machine :

Type	Size (bytes)	Range	Format specifier
char or signed char	1	– 128 to 127	%c
unsigned char	1	0 to 255	%c

4. **Void type :** Void type is usually used to specify the type of functions which returns nothing.

Que 1.32. **What do you understand by ASCII value of a character ? Can we use expressions including both integer data type and character data type ? Justify your answer.**

AKTU 2014-15(Sem-2), Marks 10

Answer

1. The acronym ASCII stands for the American Standard Code for Information Interchange.
2. The original ASCII character standard assigned a 7-bit binary number (00h-7Fh) to code 127 characters.
3. The original 7-bit 127 character ASCII code has been expanded to 8-bits (80h-FFh) to code international and graphics characters.
4. Most of the ASCII characters are the familiar symbols seen on keyboards, such as letter, numbers and punctuation marks.
5. Some of the ASCII characters are called control characters or non printable characters.
6. The ASCII code was originally designed to handle typewriter style keyboard.
7. As computer keyboards added new keys, such as function keys and cursor control keys, new key codes have been defined.

Justification : Yes, we can use expression including both integer and character data type because whenever a character constant or variable is used in an expression in C, it is automatically converted to, and subsequently treated as, an integer value.

Que 1.33. **What is variable ? How a variable is declared, defined and initalized ?**

Answer

Variable :

1. A variable is a name given to a storage area that our programs can manipulate.
2. Each variable in C language has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.
3. The name of a variable can be composed of letters, digits, and the underscore character.
4. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because C language is case-sensitive.
5. A variable in C language must be given a type, which defines what type of data the variable will hold. It can be :
   i. char : Can hold/store a character in it.
   ii. int : Used to hold an integer.
   iii. float : Used to hold a float value.
   iv. double : Used to hold a double value.
   v. void

Declaring, Defining and Initializing a variable :

1. Declaration of variables must be done before they are used in the program.
2. A variable is declared using the extern keyword, outside the main() function.
3. Defining a variable means the compiler has to now assign storage to the variable because it will be used in the program.
4. To define a function we must provide the datatype and the variable name. We can even define multiple variables of same datatype in a single line by using comma to separate them.
5. A variable can be initialized and defined in a single statement.

Example :

```
#include<stdio.h>
// Variable declaration (optional)
extern int a, b;
extern int c;
int main () {
/* variable definition: */
int a, b;
/* actual initialization */
a = 7;
b = 14;
/* using addition operator */
c = a + b;
/* display the result */
printf("Sum is : %d \n", c);
return 0;
}
```

Output :

Sum is : 21

Que 1.34. **What do you mean by scope of variable ?**

OR

Discuss scope of variables. **AKTU 2013-14(Sem-2), Marks 05**

Answer

Variable : Refer Q. 1.33, Page 1–31E, Unit-1.

Scope of variables : Scope determine the region of the program in which a defined object is visible, *i.e.*, the part of the program in which we can use the object's name.

Types of variables :

1. **Local variable :** The scope of local variable is local to block in which variable is defined. The life of the local variable is till the control remains within the block in which variable is defined.
2. **Global variable :** The scope of global variable is global *i.e.*, if a variable is declared outside, it is available to all functions that want to use it. The life of global variable remains as long as the program's execution does not come to end.
3. **Static variable :** The scope of static variable is local to the block in which the variable is defined. The life of static variable *i.e.*, value of the variable persists between different function calls.

Que 1.35. **Describe storage class.**

OR

Describe various storage classes supports in C, with suitable example. **AKTU 2013-14(Sem-1), Marks 10**

OR

What is storage class ? Describe automatic, register, static and external with neat syntax. **AKTU 2014-15(Sem-1), Marks 05**

OR

What is meant by storage classes of a variable ? Define all types of storage classes with example. **AKTU 2017-18(Sem-1), Marks 07**

Answer

The storage class is associated with a variable which decides what would be the default value of variable, where it would get stored, what will be the life of variable and what will be the scope of variable.

There are four types of storage classes available in C :

1. **Automatic storage class: (This is default class)**

Keyword	: auto
Default value	: garbage
Storage	: memory
Scope of variable	: local to block
Life of variable	: till control remains in the block in which it is declared
Declaration	: auto type-of-variable variable list

For example :

```
#include<stdio.h>
#include<conio.h>
main()
{
auto int i = 3;
{
    auto int i = 2;
    {
        auto int i = 1;
        printf("%d",i);
    }
    printf("%d", i);
}
printf("%d", i);
getch();
}
```

Output : 123

2. **Register storage class :**

Keyword	: register
Default value	: garbage
Storage	: registers of CPU
Scope of variable	: local to block
Life of variable	: till the control remains in that block where it is defined
Declaration	: register type-of-variable variable list

For example :

```
#include<stdio.h>
#include<conio.h>
main()
    {
    register int k;
    for (k = 0; k < 5; k++)
    printf("\n Hello");
    getch();
    }
```

Output :

```
Hello
Hello
Hello
Hello
Hello
```

3. **Static storage class :**

Keyword	: static
Default value	: zero
Storage	: memory
Scope variable	: local to block
Life of variable	: till program is running, and it persist in various function calls
Declarations	: static data type list of variable

For example :

```
main( )
    {
        int r( );
        int r( );
        int r( );
    }
    int r( );
    {
        static int count;
        count++;
        printf("\n%d", count);
    }
```

Output :

```
1
2
3
```

The variable count in function int r( ) is declared as static hence value of this variable is retained till the execution of program and when control reaches in function int r( ), count value is available. But it is not available outside function int r( ).

4. **External storage class :**

Keyword	: extern
Default value	: zero
Storage	: memory
Scope of variable	: global from point of declaration onwards
Life of variable	: till program execution does not come to an end

For example :

```
main( )
{
extern int i = 5;
    printf("%d", i);          *5*/
    increment( );
    dcr( );
    printf("%d", i);          /*5*/
}
increment( )
{
    i++;
    printf("%d", i);          /*6*/
}
dcr( )
{   i - - ;
    printf("%d", i);          /*5*/
}
```

When reference to the global variable occurs before their declaration, then only it is required to put declaration of that variables using keyword extern in the block in which it is referred.

If global variable is declared before all functions then no need to redeclare it using extern.

Que 1.36. Illustrate and compare the static and extern storage classes with a relevant example in C.

AKTU 2015-16(Sem-1), Marks 10

Answer

Static and extern storage classes : Refer Q. 1.35, Page 1–32E, Unit-1.

Comparison :

S.No.	Features	Static storage class	Extern storage class
1.	Lifetime	Variable retains its value between different function calls.	Throughout the program execution.
2.	Scope	Local to the block in which the variable is declared.	In all function of a program, *i.e.*, global.
3.	Storage	Memory	Memory
4.	Default initial value	Zero	Zero

5.	For example	main ( )
		{
		int r ( );
		int r( );
		int r( );
		}
		int r( );
		{
		static int count;
		count ++;
		printf("\n%d", count);
		}
			extern int i = 5 ;
			printf("%d", i);
			increment ( );
			dcr ( ) ;
			printf("%d", i);
			}
			increment ( )
			{
			i++;
			printf("%d", i);
			}
			dcr ( )
			{ i - - ;
			printf ("%d", i);
			}

Que 1.37. Write a program in C that will read a positive number from the keyboard and print it in reverse order.
For example : input : 24578 output : 87542

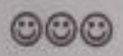
AKTU 2015-16(Sem-2), Marks 15

Answer

```
#include<stdio.h>
#include<conio.h>
main()
{
        long int rev_no;
        int n, d1, d2, d3, d4, d5;
```

```
        printf("Enter the five digit number :");
        scanf("%d",&n);
        d1 = n%10;
        n = n/10;
        d2 = n%10;
        n = n/10;
        d3 = n%10;
        n = n/10;
        d4 = n%10;
        n = n/10;
        d5 = n%10;
        rev_no = (long) d1 * 10000 + d2 * 1000 + d3 * 100 + d4 * 10 + d5;
        printf("\n Reverse number is: %ld",rev_no);
        getch();
        return 0;
}
```

Output : Enter the five digit number: 24578
Reverse number is: 87542



2 UNIT Arithmetic Expressions & Conditional Branching

PART-1

Arithmetic Expressions and Precedence : Operators and Expression using Numeric and Relational Operators, Mixed Operands, Type Conversion, Logical Operators, Bit Operator, Assignment Operator, Operator Precedence and Associativity.

CONCEPT OUTLINE : PART-1

- An operator is a symbol that tells the computer to perform certain mathematical and logical manipulations.
- Operators in C language :
  - i. Arithmetic operators
  - ii. Relational operators
  - iii. Logical operators
  - iv. Assignment operators
  - v. Increment and Decrement operator
  - vi. Conditional operator
  - vii. Bitwise operator
  - viii. Special operator
- Type conversion in C :
  - i. Implicit type conversion
  - ii. Explicit type conversion

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.1. **Write a short note on operators in C language. What are the different types of operator ?**

OR

Explain logic and bit operators with example.

AKTU 2014-15(Sem-1), Marks 10

Answer

1. An operator is a symbol that tells the computer to perform certain mathematical and logical manipulations.

2. Operators are used in programs to manipulate data and variable.
3. An operator operates on variables and performs an action in a program. It consists of words or symbols.

Different types of operators :

1. Arithmetic operators :

i. An arithmetic operator is a symbol which performs an arithmetic operation namely, addition, subtraction, etc.

ii. The data on which such operations are carried out may be a variable or a constant.

iii. C supports the arithmetic operators as given in Table 2.1.1

Table 2.1.1. Arithmetic operators

Operators	Symbol	Form	Operation
Multiplication	*	x*y	x times y
Division	/	x/y	x divided by y
Remainder	%	x%y	remainder of x divided by y
Addition	+	x + y	y is added to x
Subtraction	–	x – y	y is subtracted from x

2. Relational operator :

i. Relational operators such as greater than (>) or less than (<) are used to compare values between two variables and thus form relational expressions.

ii. C supports six relational operators in all. These operators and their meanings are shown in Table 2.1.2.

Table 2.1.2. Relational operators

Operator	Meaning
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
!=	not equal to

3. Logical operator :

i. C has the following three logical operators :

Operator	Meaning
&&	logical AND
||	logical OR
!	logical NOT

ii. The logical operators && and || are used when we want to test more than one condition and make decisions.

For example : a > b && x = = 10

An expression of this kind, which combines two or more relational expression, is termed as a logical expression or a compound relational expression.

4. **Assignment operator :**
   i. The assignment operator is represented by the equal sign (=).
   ii. The variable appearing on the left side of = sign and it is assigned the value appearing on the right side of this sign.
   iii. The assignment statement takes the following format :
   variable_name = expression ;
   x – = y ; is same as x = x – y;
   x * = y ; is same as x = x * y;
   x / = y ; is same as x = x / y ;
   x % = y ; is same as x = x % y;
5. **Increment and decrement operator :**
   i. The increment and decrement operators in C are represented by ++ and – – sign respectively.
   ii. The operator ++ means "add 1", and the operator – – means "subtract 1".
6. **Conditional operator :** A ternary operator pair "? :" is available in C to construct conditional expressions of the form :
   exp1 ? exp2 : exp3
   where exp1, exp2, and exp3 are expressions.
7. **Bitwise operators :**
   i. Bitwise operators are special operators which are used for manipulation of data at bit level.
   ii. These operators are used for testing the bits, or shifting them right or left. Bitwise operators may not be applied to float or double.

Table 2.1.3. Bitwise operators

Operator	Meaning
&	bitwise AND
|	bitwise OR
^	bitwise exclusive OR
<<	shift left
>>	shift right

For example :

```
#include<stdio.h>
main ( )
{
    unsigned int x, y;
    x = 128, y = 32 ;
    x = x >> 1 ;
    printf ("After right-shifting by 1, x = % d\n", x) ;
    y = y << 2 ;
    printf (After left-shifting by 2, y = % d\n", y) ;
}
```

Output :

After right-shifting by 1, x = 64
After left-shifting by 2, y = 128

8. **Special operators :** C supports some special operators of interest such as comma operator, sizeof operator, pointer operators (& and *) and member selection operators (. and –>).

Que 2.2. **Describe mixed mode operation.**

Answer

1. Expression that involve elements of real and integer type are called mixed mode expressions.
2. If an expression containing both integer and floating point elements is evaluated then the result will be of type float.
3. If this value is to be assigned to some identifier, then we should know the data type of the identifier.
4. If the identifier is an int and the resulting value to be assigned is float, then the floats fractional value must be truncated.
5. The operators that are used in forming mixed mode expressions are basic arithmetic operators, and the modulus operator (*i.e.*, +, –, *, / %).

For example : Consider the following program segment :

```
int a = 4, b = 2;
float x = 2, y = 3;
a = (x/y) + a/b;
```

In the expression, we have

$$a = (x/y) + a/b = (2.0/3.0) + 4/2;$$
$$= 0.6 + 2 = 2.6$$

But a is an integer quantity, so the fractional part is truncated.

i.e., $a = 2$

Que 2.3. **Discuss type conversion in C.**

OR

Discuss mixed operands and type conversions.

AKTU 2013-14(Sem-2), Marks 05

Answer

Mixed operand : Refer Q. 2.2, Page 2–5E, Unit-2.

Type conversion : Type conversion is a process of converting operators of different data type into a common data type. If we use two or more different types of data items together in an expression, the C language compiler automatically carries out the conversion of data types whenever needed.

1. **Implicit type conversion :**
   a. C permits mixing of constants and variables of different types in an expression.
   b. C automatically converts any intermediate values to the proper type so that the expression can be evaluated without losing any significance.
   c. This automatic conversion is known as implicit type conversion.
2. **Explicit type conversion :**
   a. C performs type conversion automatically. However, there are instances when we want to force a type conversion in a way that is different from the automatic conversion.
   b. Consider an example, calculation of ratio of females to males in a town.

      ratio = female_number/male_number

   c. Since female_number and male_number are declared as integers in the program, the decimal part of the result of the division would be lost and ratio would represent a wrong figure.
   d. This problem can be solved by converting locally one of the variables to the floating point as shown :

      ratio = (float) female_number/male_number

   e. The process of such a local conversion is known as explicit conversion or casting a value.

Que 2.4. **Write the difference between type conversion and type casting. What are the escape sequence characters ?**

AKTU 2014-15(Sem-2), Marks 10

AKTU 2015-16(Sem-1), Marks 7.5

Answer

Difference between type conversion and type casting :

S. No.	Type conversion	Type casting
1.	Type conversion is a process of converting operators of different data type into a common data type.	Type casting is the process by which one type of data is forcibly converted into another type.
2.	In general, only automatic conversions are those that convert a "narrower" operand into a "wider" one without losing information.	In some cases, the conversion from one type to another takes place automatically.
3. a. b.	It is done by two approaches : Implicit Explicit	It uses unary cast operators.

Escape sequences character :

1. It is a character constant enclosed between two single quotes.
2. We can also use a backslash (\) before the character. The backslash is known as the escape character.
3. It is used when the character that we want to represent does not have any graphic associated with it.
4. The escape character states that what follows is not the normal character but something else.

Table 2.4.1 : Backslash character constants

Constant	Meaning
'\a'	audible alert (bell)
'\b'	backspace
'\f'	form feed
'\n'	new line
'\r'	carriage return
'\t'	horizontal tab
'\v'	vertical tab
'\'	single quote
'\?'	question mark
'\\'	backslash
'\0'	null

Que 2.5. Write a short note on operator precedence and associativity in C language. **AKTU 2013-14(Sem-2), Marks 05**

Answer

1. C has a precedence associated with it. This precedence is used to determine how an expression involving more than one operator is evaluated.
2. There are distinct levels of precedence and an operator may belong to one of these levels. The operators at the higher level of precedence are evaluated first.
3. The operators of the same precedence are evaluated either from 'left to right' or from 'right to left', depending on the level. This is known as the associativity property of an operator.
4. Table 2.4.1 provides a complete list of operators, their precedence levels, and their rules of association. The groups are listed in the order of decreasing precedence.
5. Consider the following conditional statement :
   if (x == 10 + 15 && y < 10)
6. The precedence rules say that the addition operator has a higher priority than the logical operator (&&) and the relational operators (== and ,). Therefore, the addition of 10 and 15 is executed first. This is equivalent to :
   if (x = = 25 && y < 10)

Table 2.5.1.

Description	Operator	Associativity
Function expression	( )	Left to right
Array expression	[ ]	Left to right
Structure operator	->	Left to right
Structure operator	.	Left to right
Unary minus	–	Right to left
Increment/Decrement	++ / – –	Right to left
One's complement	~	Right to left
Negation	!	Right to left
Address of	&	Right to left
Value at address	*	Right to left
Type cast	(type)	Right to left
Size in bytes	sizeof	Right to left
Multiplication	*	Left to right
Division	/	Left to right

Modulus	%	Left to right
Addition	+	Left to right
Subtraction	–	Left to right
Left shift	<<	Left to right
Right shift	>>	Left to right
Less than	<	Left to right
Less than or equal to	<=	Left to right
Greater than	>	Left to right
Greater than or equal to	>=	Left to right
Equal to	==	Left to right
Not equal to	!=	Left to right
Bitwise AND	&	Left to right
Bitwise exclusive OR	^	Left to right
Bitwise inclusive OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	? :	Right to left
Assignment	=	Right to left
	*= /= %=	Right to left
	+= -= &=	Right to left
	^= |=	Right to left
	<<= >>=	Right to left
Comma	,	Right to left

Que 2.6. What are the different types of operators in C language and also write down the difference between the associativity and precedence of operators. **AKTU 2014-15(Sem-2), Marks 10** **AKTU 2015-16(Sem-1), Marks 7.5**

Answer

Different types of operators : Refer Q. 2.1, Page 2–2E, Unit-2.

Difference between associativity and precedence of operators :

S. No.	Associativity of operators	Precedence of operators
1.	Associativity decides which operator to execute first when there is more than one operator with same priority.	Precedence decides which operator to execute first when there is more than one different operator with different priorities.
2.	Associativity indicates in which order two operators of same precedence (priority) executes.	If more than one operator is involved in an expression then, C language has predefined rule of priority of operators. This rule of priority of operators is called operator precedence.

Que 2.7. **Write a program in C to find greatest number among three numbers using conditional operator.**

AKTU 2013-14(Sem-1), Marks 05

Answer

```
#include<stdio.h>
#include<conio.h>
main( )
{
int a, b, c, big ;
printf("enter three numbers a, b, c") ;
scanf("%d %d%d", &a, &b, &c) ;
big = (a > b) ? ((a > c) ? (a : c)) : ((b < c) ? b : c)) ;
printf("% The largest number is : % d", big) ;
getch( ) ;
}
```

PART-2

Conditional Branching : Applying if and Switch Statements, Nesting if and else, Use of Break and Default with Switch.

CONCEPT OUTLINE : PART-2

- The if statement is a control statement that tests a particular condition.
- The if-else statement is an extension of the simple if statement.
- When a series of decisions are involved, we have to use more than one if else statement in nested form *i.e.*, nested if else statement.
- The break statement transfers the control out of the switch statement.
- The default label is optional. If present, it will be executed when the expression does not find a matching case label.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.8. **Describe if statement. Give its flowchart.**

Answer

1. If statement is a control statement that test a particular condition. Whenever, the evaluated condition comes out to be true, then that action or the set of actions are carried out. Otherwise, the given set of action(s) are ignored.
2. The general form of a simple if statement is :

```
if (test expression)
{
statement-block;
}
statement-x;
```

3. The 'statement-block' may be a single statement or a group of statement.
4. If the test expression is true, the statement-block will be executed; otherwise the statement-block will be skipped and the execution will jump to the statement-x.
5. When the condition is true both the statement-block and the statement-x are executed in sequence. This is illustrated in Fig. 2.8.1

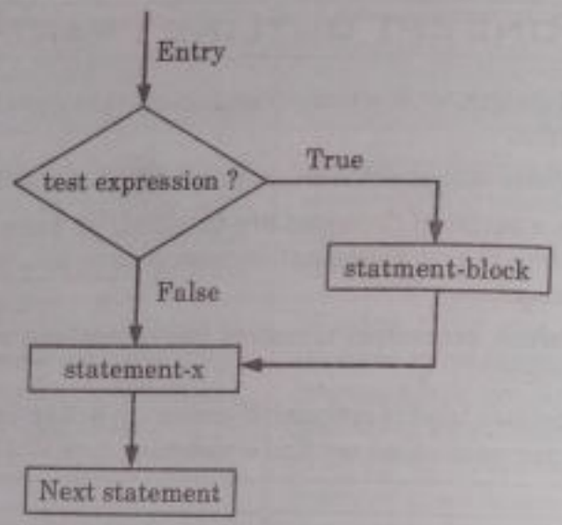


Fig. 2.8.1. Flow chart of simple if control.

Que 2.9. Explain if-else statement. Give its syntax.

Answer

1. The if-else statement is an extension of the simple if statement. The general form is :

```
If (test expression)
{
True-block statement (s)
}
else
{
False-block statement (s)
}
statement-x
```

2. If the test expression is true, then the true-block statement(s), immediately following the if statement are executed; otherwise, the false-block statement(s) are executed.
3. In either case, either true-block or false-block will be executed, not both. This is illustrated in Fig. 2.9.1.
4. In both the cases, the control is transferred subsequently of the statement-x.

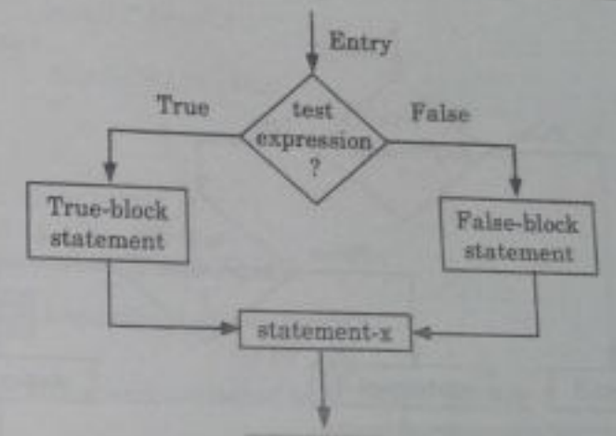
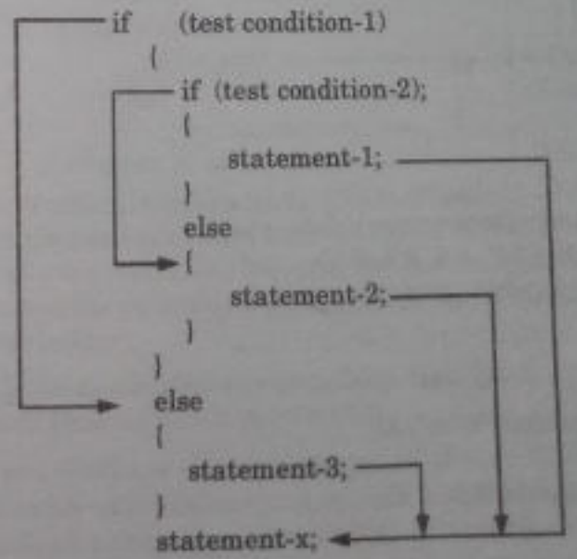


Fig. 2.9.1.

Que 2.10. What is nesting of if-else statement ? Give its flowchart.

Answer

1. When a series of decisions are involved, we may have to use more than one if-else statement in nested form as shown in Fig. 2.10.1.
2. If the condition-1 is false, the statement-3 will be executed; otherwise it continues to perform the second test.
3. If the condition-2 is true, the statement-1 will be evaluated; otherwise the statement-2 will be evaluated and then the control is transferred to the statement-x.



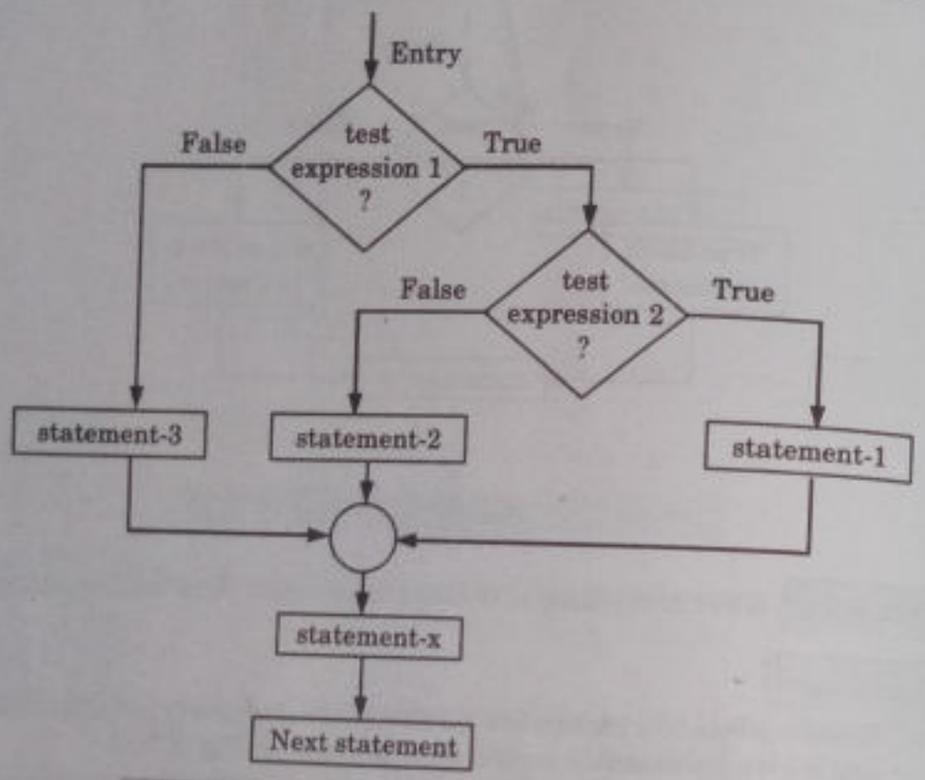


Fig. 2.10.1. Flow chart of nested if-else statement.

Que 2.11. **Write a program to print the largest of three number using nested if-else statements.**

Answer

```
#include<stdio.h>
#include<conio.h>
    main ( )
    {
    float A, B, C;
    printf ("Enter three values\n");
    scanf ("%f %f %f", &A, &B, &C);
    printf ("\nLargest value is");
    if (A > B)
    {
        if (A > C)
            printf ("%f\n", A);
        else
            printf ("%f\n", C);
    }
    else
    {
        if (C > B)
            printf ("%d\n", C);
        else
            printf ("%d\n", B);
    }
    }
```


Output :

```
Enter three values
23445   67379   88843
Largest value is 88843.000000
```

Que 2.12. **Describe if-else-if ladder.**

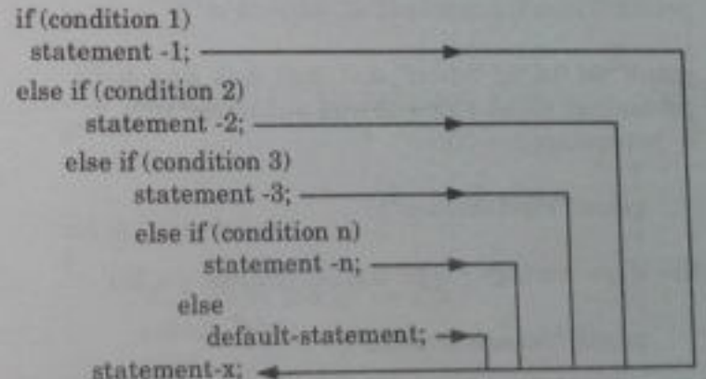
OR

What are the disadvantages of if-else-if ladder ?

AKTU 2015-16(Sem-2), Marks 10

Answer

1. There is an another way of putting if's together when multipath decisions are involved.
2. A multipath decision is a chain of if's in which the statement associated with each else is an if. It takes the following general form :



   This construction is known as the if-else-if ladder.
3. The conditions are evaluated from the top (of the ladder), to downwards. As soon as a true condition is found, the statement associated with it is executed and the control is transferred to the statement-x (skipping the rest of the ladder).
4. When all the *n* condition becomes false, then the final else containing the default statement will be executed.

Following are various disadvantages of if-else-if ladder :

1. As the number of conditions goes on increasing, the level of zig-zag form of if-else ladder also goes on increasing. As a result, the whole program creeps to the right.

2. Care needs to be exercised to match the corresponding if and else.
3. Care needs to be exercised to match the corresponding pair of braces.
4. When nested-if increase, zig-zag form of if-else ladder increases as well as it becomes difficult to understand the program.

Que 2.13. **The marks obtained by a student in five different subjects are input through the keyboard. The student gets a division as per the following rules :**

a. **Percentage above or equal to 60 : First division**
b. **Percentage between 50 and 59 : Second division**
c. **Percentage between 40 and 49 : Third division**
d. **Percentage less than 40 : Fail**

Write a program to calculate the division obtained by the student.

Answer

```
#include<stdio.h>
#include<conio.h>
void main( )
{
int s1, s2, s3, s4, s5;
float percentage;
clrscr( );
printf("Enter the marks of all subjects\n");
}
scanf("%d %d %d %d %d", &s1, &s2, &s3, &s4, &s5);
percentage = ((s1 + s2 + s3 + s4 + s5)/500)*100;
if (percentage > = 60)
    {
    printf("First division");
    }
else if (percentage > = 50 && percentage < = 59)
    {
    printf("Second division");
    }
else if (percentage > = 40 && percentage < = 49)
    {
    printf("Third division");
    }
else
    {
    printf("Fail");
    }
getch( );
}
```


Que 2.14. **Write a program to check whether the given character is in upper case, lower case or non alphabetic character.**

AKTU 2015-16(Sem-2), Marks 10

Answer

```
#include<stdio.h>
#include<conio.h>
int is_low(char);
int is_alpha(char);
    main()
    {
        char ch;
        printf("Enter a character:");
        scanf("%c", &ch);
if (is_alpha(ch))
        {
        printf("The character is alphabetic \n");
        if(is_low(ch))
            printf("The character is of lower case \n");
        else
            printf("The character is of upper case \n");
        }
        else
            printf("The character is non-alphabetic \n");
        getch( );
        return 0;
}
        int is_low(char ch)
        {
            if(ch >= 97 && ch <= 122 )
            return (1);
            else
            return (0);
        }
        int is_alpha(char ch)
        {
if(ch >= 'a' && ch <= 'z') | |(ch >='A' && ch <= 'Z'))
        {
        return (1);
        }
        return (0);
        }
```

Output :

```
Enter a character : g
The character is alphabetic
The character is of lower case
```

Que 2.15. **Any year is input through the keyboard. Write a program to determine whether the year is a leap year or not.**

AKTU 2016-17(Sem-1), Marks 05

Answer

```
#include<stdio.h>
#include<conio.h>
main()
{
        int yr;
        printf("\nEnter any year:");
        scanf("%d",&yr);
        if((yr%400 == 0)||(yr%100 != 0 && yr%4 == 0))
        printf("\n%d is a leap year.",yr);
        else
        printf("\n%d is not a leap year.",yr);
        getch();
        return 0;
}
```

Output :
```
Enter any year: 2012
2012 is a leap year.
Enter any year: 2013
2013 is not a leap year.
```

Que 2.16. **A company gives insurance to its drivers in the following cases :**

i. **If the driver is married.**
ii. **If the driver is unmarried, male & above 30 years of age.**
iii. **If the driver is unmarried, female & above 25 years of age.**

In all other cases, the driver is not insured. If the marital status, gender and age of a driver are inputs, write a program to determine whether the driver is to be insured or not.

AKTU 2016-17(Sem-2), Marks 07

Answer

```
#include<stdio.h>
main()
{
```



```
char sex,ms;
int age;
printf ("Enter age, sex, marital status:");
scanf ("%d %c %c", &age, &sex, &ms);
if (ms=='M')
printf ("The driver is insured");
else
{
if (sex=='M')
{
if (age>30)
printf ("Driver is insured");
else
printf ("Driver is not insured");
}
else
{
if (age>25)
printf ("Driver is insured");
else
printf ("Driver is not insured");
}
}
}
```

Que 2.17. **A five digit number is entered through the keyboard. Write a program to obtain the reversed number and to determine whether the original and reversed numbers are equal or not.**

AKTU 2016-17(Sem-2), Marks 05

Answer

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n,a,b ;
long int revnum = 0;
```

```
clrscr();
printf("\n Enter a five digit number (less than 32767) :");
scanf("%d", &n);
a=n%10;
n=n/10;
revnum = revnum + a * 10000;
a=n%10;
n=n/10;
revnum = revnum + a * 1000;
a=n%10;
n=n/10;
revnum = revnum + a * 100;
a=n%10;
n=n/10;
revnum = revnum + a * 10;
a=n%10;
revnum = revnum + a;
printf("The reversed number is % ld", revnum);
If (n==revnum)
{
printf ("Numbers are equal");
}
else
{
printf ("Numbers are not equal");
}
getch();
}
```

Que 2.18. **Explain switch statement.**

OR

What is the role of switch statement in C programming language ? Explain with example.

AKTU 2014-15(Sem-2), Marks 10

AKTU 2016-17(Sem-1), Marks 05

Answer

1. Switch case rtatement is a multiple branch selection statement, which successively tests the value of an expression against a list of integer or constant.

2. It allows us to choose a block of statements among several alternatives.
3. The switch statement is often used to process keyword commands such as menu selection.
4. The switch statement is especially useful when the selection is based on the value of single variable or of a simple expression. The value of this expression may be of type int or char, but not of type double.
5. **General form of switch statement / Structure of switch statement is :**

```
switch (expression)
{
case constant 1:
statement sequence;
break;
case constant 2:
statement sequence;
break;
:
:
default:
statement sequence;
}
```

For example :

```
#include<stdio.h>
#include<conio.h>
int main( )
int a = 2, b = 5, x = 3, N, Y;
printf("Enter the value of N : \n");
scanf("%d", &N);
switch(N)
{
case 1:
Y = (a * x + b) ^ 2;
printf("The value of Y = %d", Y);
break;
case 2:
Y =(a * x ^ 2) + (b ^ 3);
printf("The value of Y = %d", Y);
break;
case 3:
Y = –a * x + b;
printf("The value of Y = %d", Y);
break;
case 4:
Y = a ^ 2 + x;
printf("The value of Y = %d", Y);
break;
default:
```

```
printf("This is default");
}
return 0;
getch();
}
```

Que 2.19. Describe the function of default and break in switch statement.

Answer

Function of break statement :

1. The break statement transfers the control out of the switch statement.
2. The break statement is optional. That is, two or more case labels may belong to the same statement.

Function of default statement :

1. The default label is optional. If present, it will be executed when the expression does not find a matching case label.
2. There can be at most one default label.
3. The default may be placed anywhere but usually placed at the end.
4. It is permitted to nest switch statements.

Que 2.20. Differentiate between nested-if and switch statements in C with example. **AKTU 2013-14(Sem-1), Marks 05**

Answer

Difference between nested-if and switch statements :

S. No.	Nested-if	Switch statements
1.	Nested-if checks every condition.	Switch statement checks the condition first and jumps to the suitable case statement.
2.	In nested-if execution time will be more and code becomes lengthy.	A switch statement is more structured. Hence, execution time is less.

3.	The syntax of nested-if is as follows :
	if (condition 1)
	{
	if (condition 2)
	{
	statement 1;
	}
		The syntax of switch statement is as follows :
		switch (expression)
		{
		case value 1 :
		statement ;
		statement ;
		break ;

	else
	{
	statement 2;
	}
		=====
		default :
		statement ;
		statement ;

Example of nested-if statement : Refer Q. 2.11, Page 2–14E, Unit-2.
Example of switch statement : Refer Q. 2.18, Page 2–20E, Unit-2.

Que 2.21. Write a program to print all prime numbers from 1 to 300. **AKTU 2016-17(Sem-1), Marks 05**

Answer

```
#include<stdio.h>
#include<conio.h>
void main( )
{
int num, i, flag;
clrscr( );
printf("Prime numbers between 1 to 300 are :");
for(num = 2; num < 300; num++)
        {
        flag = 0;
        for(i = 2; i < num/2; i ++)
        {
        if(num%i = = 0)
                {
                flag = 1;
                break;
                }
        }
                if(flag = = 0)
                printf("%d\n", num);
        }
        getch( );
}
```

Que 2.22. Write a program in C to display the prime numbers between 1 and 100. **AKTU 2016-17(Sem-2), Marks 05**

Answer

Refer Q. 2.21, Page 2–23E, Unit-2.
Since, the display range is changed from 1 to 100
Therefore changing the for condition *i.e.*, for (num = 2 ; num < 100 ; num ++)
Rest program is same

Que 2.23. **Explain preprocessor.**

OR

Explain the following :

i. **Preprocessor**

ii. **Conditional operators** **AKTU 2013-14(Sem-1), Marks 10**

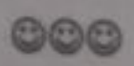
Answer

i. **The standard C preprocessor :**

1. The C preprocessor is a collection of special statements, called directives, which are executed at the beginning of the compilation process.
2. The C preprocessor provides several tools that are unavailable in other high level languages.
3. We can include various instructions to the compiler in the source code of a C program. These are called preprocessor directives.
4. The C preprocessor is a simple macro processor that conceptually processes the source text of a C program before the compiler properly reads the source program.
5. The preprocessor is controlled by special preprocessor command lines, which are lines of source file beginning with the character #.
6. The syntax of preprocessor commands is completely independent of the syntax of the rest of the C language.
7. Preprocessor operates under the control of what is known as preprocessor command lines or directives. The preprocessor directives are :

Command	Meaning
#define	Define a preprocessor macro
#undef	Remove a preprocessor macro definition
#include	Insert text from another source file
#if	Conditionally include some text based on the value of a constant expression
#ifdef	Conditionally include some text based on whether a macro name is defined
#ifndef	Conditionally include some text with the sense of the test opposite to that of #ifdef.
#else	Alternatively include some text if the previous #if, #ifdef, #ifndef, or #else test failed.

ii. **Conditional operators :** Refer Q. 2.1, Page 2-2E, Unit-2.



3 UNIT

Loops and Functions

PART-1

Iteration and Loops : Use of While, Do-While and For Loops, Multiple Loops Variables, Use of Break and Continue Statements.

CONCEPT OUTLINE : PART-1

- A control structure that repeats a group of steps in a program is called a loop.
- Three kinds of loop control statement are :
  i. While statement
  ii. Do-while statement
  iii. For statement

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.1. What is loop ? Discuss the different types of loop.

OR

Describe about the types of looping statements in C with necessary syntax. **AKTU 2014-15(Sem-1), Marks 10**

Answer

1. A loop in a program consists of two parts, one is called the body of the loop and the other is called the control statement.
2. The control statement performs a logical test whose result is either true or false.
3. If the result of the logical test is true, then the statements contained in the body of the loop are executed. Otherwise, the loop is terminated.
4. C provides the following three types of loop control statements :

a. While statement :

i. While statement is used to execute a set of statements repeatedly, as long as the specified condition is true.

ii. The syntax of a while loop is as follows :

```
while (logexp)
        {
          statement ;
        }
```


iii. The flow chart of while statement is shown in Fig. 3.1.1.

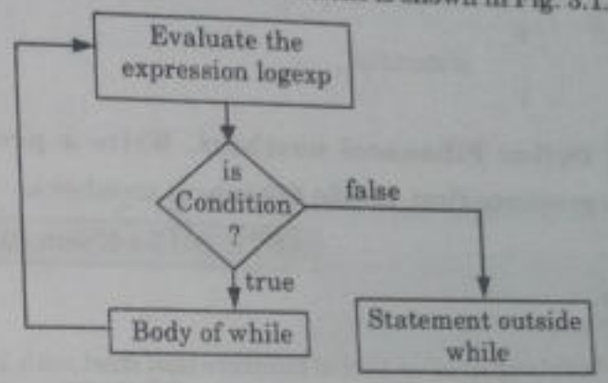


Fig. 3.1.1. Flow chart of while statement.

b. Do-while statement :

i. This is used to execute a set of statements repeatedly, until the logical test results in false. This is called the post-test loop, because, the test for repetition is made at the end of each pass.

ii. The syntax of do-while is as follows :

```
do
{
        statement ;
}
while (logexp) ;
```

iii. The flow chart of do-while statement is shown in Fig. 3.1.2.

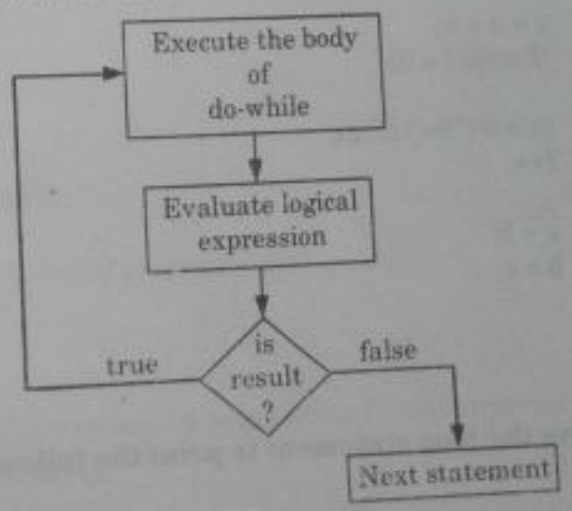


Fig. 3.1.2.

c. For statement :

i. For statement is used when the programmer knows how many times a set of statements are to be executed.

ii. The syntax of a for statement is as follows :

```
for (expression1; expression2; expression3)
{
    statement ;
}
```

Que 3.2. **Define Fibonacci numbers. Write a program in C language to generate first 10 odd Fibonacci numbers.**

AKTU 2013-14(Sem-2), Marks 05

Answer

Fibonacci numbers (sequence) is a set of numbers that start with 1 or 0 followed by a 1 proceeds on the rule that each number *i.e.*, Fibonacci number is equal to the sum of the preceding two numbers.

$F(n) = 0, 1, 1, 2, 3, 5, 8, 13$ .....

Program to find first 10 odd Fibonacci numbers :

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a = 0, b = 1, c ;
    int f = 2;
    clrscr ();
    printf ("%d\n%d", a, b);
    while (f < 10)
{
            c = a + b;
            if((c%2)! = 0)
    {
            printf("%d\n", c);
            f++;
    }
            a = b;
            b = c;
}
getch();
}
```

Que 3.3. **Give the loop statement to print the following sequence of integer.**

-6-4-20246

AKTU 2014-15(Sem-2), Marks 10

Answer

We use for loop statement to print the given sequence of integers. For loop statement is given as :


```
for (i = – 6; i < 7; i = i + 2)
{
printf("%d", i);
}
```

Que 3.4. **Write a program in C language to generate the Fibonacci series.**

AKTU 2014-15(Sem-2), Marks 10

AKTU 2016-17(Sem-1), Marks 05

Answer

```
#include<stdio.h>
#include<conio.h>
main()
{
int prev = 0;
int next = 1;
int f = 1;
int i, n;
printf("Enter the value of n to generate Fibonacci series:");
scanf("%d",&n);
printf("\n The Fibonacci series is : \n");
printf("\n%d",f);
for(i = 1; i <= n; i++)
{
    f = prev + next;
    prev = next;
    next = f;
    printf("\n%d",f);
}
    printf("\n\nEnd of program\n");
    getch();
    return 0;
}
```

Output : Enter the value of *n* to generate Fibonacci series : 5

The Fibonacci series is :

1

1

2

3

5

8

End of program

Que 3.5. **Write a C program to check whether a given square matrix is symmetric or not.** **AKTU 2015-16(Sem-2), Marks 05**

Answer

```
#include<stdio.h>
#include<conio.h>
void main( )
{
int m, n, i, j, a[10][10], b[10][10], x = 0;
clrscr( );
printf("Enter the number of row:");
scanf("%d", &m);
printf("Enter the number of column:");
scanf("%d", &n);
printf("Enter the matrix:\n");
for (i = 0; i < m; i++)
{
for (j = 0; j < n; j++)
{
scanf("%d", &a[i][j]);
}
printf("\n");
}
for (i = 0; i < m; i++)
{
for (j = 0; j < n; j++)
{
b[j][i] = a[i][j];
}
}
for (i = 0; i < n; i++)
{
for (j = 0; j < m; j++)
{
if (a[i][j]! = b[i][j])
{
x++;
}
}
}
printf("The matrix is =>");
x = = 0? printf("symmetric"): printf("not symmetric");
getch( );
}
```

Output :


```
Enter the number of row : 2
Enter the number of column : 2
Enter the matrix :
4
4
4
4
The matrix is => symmetric
```

Que 3.6. **Write a program in C to print the following pattern :**

```
A B C D E F G F E D C B A
A B C D E F   F E D C B A
A B C D E       E D C B A
A B C D           D C B A
A B C               C B A
A B                   B A
A                       A
```

AKTU 2015-16(Sem-1), Marks 10

Answer

```
#include<stdio.h>
#include<conio.h>
int main()
{
        int i, j, k;
        for(i = 71; i >= 65; i– –)
        {
        for(j = 65; j <= i; j++)
        {
        printf("%c", j);
        }
        for(k = 71; k > j – 1; k– –)
        {
                printf(" ");
        }
        for(k = i; k < 70; k++)
                printf(" ");
        for(j = j – 1; j >= 65; j– –)
        {
                if(j == 71)
                {
                j = 70;
                printf("%c", j);
                }
                else
```

```
            printf("%c",j);
            }
            printf("\n");
            }
    getch();
    }
```

Que 3.7. **Write a program to print the following pattern :**

A
AB
ABC
ABCD
ABCDE
ABCDEF

AKTU 2016-17(Sem-1), Marks 05

Answer

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i, j;
for(i = 1; i <= 6; i++)
{
for(j=1; j <= i; j++)
{
printf("%c",'A' + j – 1);
}
printf("\n");
}
getch();
}
```

Que 3.8. **Write a program to check whether a given number is Armstrong or not. Like $153 = 1^3 + 5^3 + 3^3$.**

AKTU 2016-17(Sem-1), Marks 05

Answer

```
#include<stdio.h>
int main(){
int num,r,sum=0,temp;
printf("Enter a number: ");
scanf("%d",&num);
temp=num;
while(num!=0){
r=num%10;
```



```
num=num/10;
sum=sum+(r*r*r);
}
if(sum==temp)
printf("%d is an Armstrong number",temp);
else
printf("%d is not an Armstrong number",temp);
return 0;
}
```

Output :
Enter a number: 153
153 is an Armstrong number

Que 3.9. **Write a program to sum the series $\frac{1}{2}+\frac{2}{3}+\frac{3}{4}+$ ..... $+\frac{n}{n+1}$.**

AKTU 2016-17(Sem-2), Marks 07

Answer

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
        int n;
        float sum = 0.0, a, i;
        clrscr ( );
        printf("\nEnter the value of n :");
        scanf("%d", &n);
        for (i = 1.0; i < n; i ++)
        {
                a = i/(i + 1);
                sum = sum + a;
        }
        printf("\n The sum of series 1/2 + 2/3 + ..... + %d/%d is % f",
        n, n + 1, sum);
    getch ( );
    }
```

Que 3.10. **Write a C program to add first seven terms of the following series using for loop.**

1/1! + 2/2! + 3/3! + ...

AKTU 2017-18(Sem-1), Marks 07

Answer

```
#include<stdio.h>
main()
{
```

```
int i;
double fact = 1,sum = 0,div;
clrscr();
for(i = 1; i <= 7; i++)
{
fact *= i;
div = i / fact;
sum += div;
}
printf("\nSum of First Seven Terms 1/1! + 2/2! + ..... + %d/%d! = %lf",
i, i, sum);
getch();
}
```

Output :

Sum of First Seven Terms : 1/1! + 2/2! + ... + 7/7! = 2.71667

Que 3.11. **Write a program to check the number is palindrome or not. The program should accept any arbitrary number typed by user.**

AKTU 2017-18(Sem-1), Marks 07

Answer

```
#include<stdio.h>
int main()
{
int n, reverse = 0, t;
printf("Enter a number :\n");
scanf("%d", &n);
t = n;
while (t != 0)
{
reverse = reverse * 10;
reverse = reverse + t%10;
t = t/10;
}
if (n == reverse)
printf("%d is a palindrome number.\n", n);
else
printf("%d is not a palindrome number.\n", n);
return 0;
}
```

Que 3.12. **Write a program to check the number is prime number or not.**

AKTU 2017-18(Sem-1), Marks 07

Answer

```
#include<stdio.h>
main() {
int n, i, count = 0;
printf("Enter any number n:");
scanf("%d", &n);
for (i = 1; i <= n; i++) {
if (n % i == 0) {
count ++;
}
}
if (count == 2) {
printf("n is a Prime number");
}
else {
printf("n is not a Prime number");
}
return 0;
}
```

Que 3.13. **Write a program in C to print the following pattern :**

```
1
2   3
4   5   6
7   8   9   10
```

AKTU 2013-14(Sem-1), Marks 05

Answer

```
#include<stdio.h>
#include<conio.h>
void main( )
        {
        int i, j, k = 1;
        clrscr( );
        for (i = 1; i < 5; i ++)
        {
        for (j = 1; j < = i; j ++)
            {
            printf("%d", k);
            k ++;
            }
        printf("\n");
    }
    getch( );
    }
```

Que 3.14. What is the purpose of the break statements ? Within which control statements can the break statement be included ?

Answer

1. The break statement is used to terminate loops or to exit from a switch statement. It can be used within for, while, do-while or switch statement.
2. The general form of the break statement is :

```
break;
```

3. A break used in a switch statement will affect only that switch.
4. In a loop statement when break is encountered, the loop is immediately terminated, and program control resumes at the next statement following the loop.

For example :

```
#include<stdio.h>
#include<conio.h>
int main
{
int a;
for(a = 0; a < 100; a++)
{
printf("%d", a);
if(a == 10)
break;
}
getch();
return 0;
}
```

Output : 012345678910

a. The above program prints the number 0 to 10 on the screen.

b. Then the loop terminates because break causes immediate exit from the loop, overriding the conditional test a < 100.

Que 3.15. What is the purpose of continue statement ? Give example and its flowchart.

Answer

1. The continue statement is used for the next iteration of the loop to take place, skipping any code in between. Hence, it is used to terminate the current iteration and continue with the next iteration of the loop.
2. For the loop, continue causes the updation and then the conditional test portions of the loop to execute. For the while and do-while loop, program control passes to the conditional tests.
3. The flowchart of the continue statement is shown in Fig. 3.15.1.

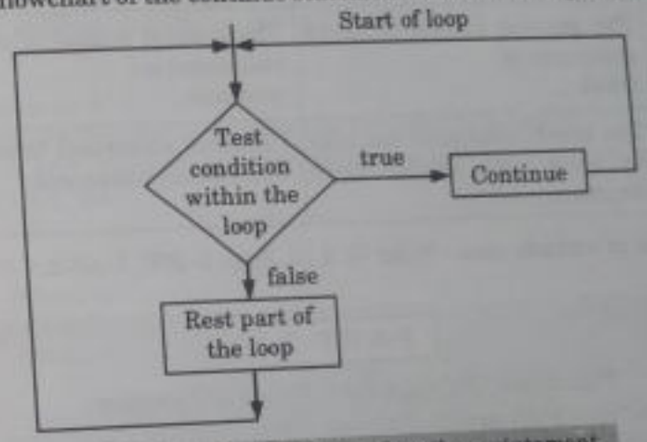


Fig. 3.15.1. Flowchart of continue statement.

4. For example,

```
for (i = 1; i <= 2; i++)
{
    for (j = 1; j <= 2; j++)
    {
        if (i == j)
            continue;
        printf ("\n%d %d", i, j)
    }
}
```

Output

```
12
21
```

Que 3.16. What is the difference between break and continue statement in C ? Describe the structure of switch-case with neat example. AKTU 2014-15(Sem-1), Marks 10

Answer

Difference between break and continue statement in C :

S.No.	Break statement	Continue statement
1.	A break statement allows us to terminate the execution of loop and to jump out of the loop.	The continue statement is used for the next iteration of the loop to take place skipping any code in between.
2.	The general format of break statement is : break ;	The general format of continue statement is : continue ;
3.	The break statement can also be used with switch case structure.	Continue statement is not used with switch statement.

Structure of switch case : Refer Q. 2.18, Page 2–20E, Unit-2.

PART-2

Functions : Introduction, Types of Functions, Functions with Array, Passing Parameters to Functions, Call by Value, Call by Reference, Recursive Functions.

CONCEPT OUTLINE : PART-2

- A function is a set of statement that takes input, do some specific computation and produces output.
- Two types of function :
  i. Standard library functions ii. User-defined function
- Two methods to pass parameter are :
  i. Call by value ii. Call by reference
- A recursive function is a function that calls itself to perform a specific operation.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.17. Discuss function with example. What are the benefits of using function ?

Answer

1. A function is a set of statements that take inputs, do some specific computation and produces output.
2. The idea is to put some commonly or repeatedly done task together and make a function, so that instead of writing the same code again and again for different inputs, we can call the function.

Example :

```
#include <stdio.h>
// An example function that takes two parameters 'x' and 'y'
// as input and returns max of two input numbers
int max(int x, int y)
{
    if (x > y)
        return x;
    eise
        return y;
}
// main function that doesn't receive any parameter and
// returns integer.
int main(void)
{
    int a = 10, b = 20;
// Calling above function to find max of 'a' and 'b'
    int m = max(a, b);
    printf("m is %d", m);
    return 0;
}
```

Output :

m is 20

Benefits of using functions :

1. It provides modularity to our program's structure.
2. It makes our code reusable. We just have to call the function by its name to use it, wherever required.
3. In case of large programs with thousands of code lines, debugging and editing becomes easier if we use functions.
4. It makes the program more readable and easy to understand.

Que 3.18. How function is declared ?

Answer

1. General syntax for function declaration is, returntype functionName(type1 parameter1, type2 parameter2,...);
2. Like any variable or an array, a function must also be declared before it is used.
3. Function declaration informs the compiler about the function name, parameters it accepts, and its return type.
4. The actual body of the function can be defined separately. It is also called as function prototyping.
5. **Function declaration consists of following four parts :**

   a. returntype :

   i. When a function is declared to perform some sort of calculation or any operation and is expected to provide with some result at the end, in such cases, a return statement is added at the end of function body.

   ii. returntype specifies the type of value(int, float, char, double) that function is expected to return to the program which called the function.

   b. functionName :

   i. functionName is an identifier and it specifies the name of the function.

   ii. The functionName is any valid C identifier and therefore must follow the same naming rules like other variables in C language.

   c. Parameter list :

   i. The parameter list declares the type and number of arguments that the function expects when it is called.

   ii. The parameters in the parameter list receive the argument values when the function is called. They are often referred as formal parameters.

   d. Terminating semicolon : The semicolon is a statement terminator in C to help the parser figure out where the statement ends.

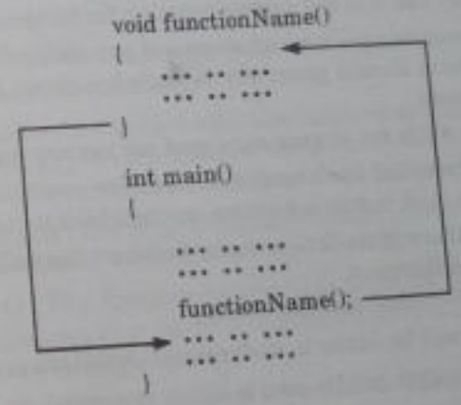
Que 3.19. What are the types of functions ?

Answer

There are two types of functions in C programming :

1. **Standard library functions (built-in functions) :**
   a. The standard library functions are built-in functions in C programming to handle tasks such as mathematical computations, I/O processing, string handling etc.

   b. These functions are defined in the header file. When we include the header file, these functions are available for use.
   c. For example, printf() is a standard library function to send formatted output to the screen (display output on the screen). This function is defined in "stdio.h" header file.
   d. There are other numerous library functions defined under "stdio.h", such as scanf(), printf(), getchar() etc. Once we include "stdio.h" in our program, all these functions are available for use.
2. **User-defined functions :**
   a. The functions created by the user are called user-defined functions.
   b. Depending upon the complexity and requirement of the program, we can create as many user-defined functions as we want.
   c. The execution of a C program begins from the main() function.
   d. When the compiler encounters functionName(); inside the main function, control of the program jumps to void functionName() as shown :



Que 3.20. What are the types of user-defined functions ? Give advantages of user-defined function.

Answer

A user-defined function, depending on whether arguments are present or not and whether a value is returned or not, may belong to one of the following categories :

1. **Function with arguments but no return value :**
   a. Here the called function receives the data from the calling function.
   b. The arguments and parameters should match in number, data type and order.

c. But, the called function does not return any value back to the calling function. Instead, it prints the data in its scope only.

d. It is a one way data communication between the calling function and the called function.

2. Function with no arguments and return value :

a. Here, the called function does not receive any data from the calling function.

b. It manages with its local data to carry out the specified task.

c. However, after the specified processing, the called function returns the computed data to the calling function.

d. It is also a one way data communication between the calling function and the called function.

3. Functions with arguments and one return value :

a. The function value receives data from the calling function through arguments, but does not send back any value.

b. Rather, it displays the results of calculation at the terminal. However, we may not always wish to have the result of a function displayed.

c. We may use it in the calling function for further processing.

d. Moreover, to assure a high degree of portability between programs, a function should generally be coded without involving any I/O operations.

4. Functions with no arguments and no return value :

a. Here, the called function does not receive any data from the calling function. And, it does not return any data back to the calling function.

b. Hence, there is no data transfer between the calling function and the called function.

Advantages of user-defined function :

1. The program will be easier to understand, maintain and debug.
2. Reusable codes that can be used in other programs.
3. A large program can be divided into smaller modules. Hence, a large project can be divided among many programmers.

Que 3.21. Write about the formatted and unformatted input/output function in C. **AKTU 2014-15(Sem-1), Marks 10**

Answer

Formatted I/O function : In order to write a user interactive program in C language, we would need input and output functions that are also called routines. The two functions used for this purpose are :

a. The printf() function :

1. The printf() function is an important function to show the formatted output on the standard output device such as monitor screen.
2. The general form of printf() function is :

   printf(" format strings", argument list) ;

b. The scanf() function :

1. The scanf() function does the opposite of printf() function. It reads data entered from the keyboard and sends it to the memory of the CPU.
2. It will read only till the next white space character (*i.e.*, '\n' or '\t' or ' ') and thus can be used to read only one word as a string should be highlighted.

Unformatted I/O functions : These functions deal with a single character or with a string of characters.

Character input : Three input functions available in most C language compilers are :

a. Getchar() : This function reads one character from the keyboard after the new line character is received (when we press enter key.)

b. getch() and getche() :

1. These two functions are very similar, as they respond without pressing the enter key.
2. The difference is that with the function getche(), the echo of the pressed key is displayed on the screen (the letter "e" stands for "echo"), but with the function getch(), there is no echoing.

Character output : The programming language C contains functions which can handle characters, while processing a file. These functions are :

a. Putchar() : The function putchar() (stands for "put character") uses single argument, that can be a character variable or the character itself but enclosed in single quotes.

b. Putc() : The function putc() is intended for files, but its output can be redirected to any standard device like the screen (stdout) or the printer (stdprn). The function putc() takes the following form :

putc (a, device);

Que 3.22. How to pass array to functions ?

Answer

Whenever we need to pass a list of elements as argument to any function in C language, it is done by passing an array to function.

This can be done in following ways :

1. Passing arrays as parameter to function :

a. In this, we will pass a single array element as argument to a function.

For example :

```
#include<stdio.h>
void giveMeArray(int a);
int main()
{
    int myArray[] = { 2, 3, 4 };
    giveMeArray(myArray[2]);
                        //Passing array element myArray[2] only.
    return 0;
}
void giveMeArray(int a)
{
    printf("%d", a);
}
```

Output :

4

2. Passing a complete one-dimensional array to a function :

a. To pass one-dimensional array to a called function, it is necessary to list the name of the array, without any subscripts and the size of array as arguments.

For example :

```
#include<stdio.h>
float findAverage(int marks[]);
int main()
{
    float avg;
    int marks[] = {99, 90, 96, 93, 95};
    avg = findAverage(marks);
                        // name of the array is passed as argument.
    printf("Average marks = %.1f", avg);
    return 0;
}
float findAverage(int marks[])
    {
        int i, sum = 0;
        float avg;
        for (i = 0; i <= 4; i++) {
        sum += age[i];
    }
    avg = (sum / 5);
    return avg;
}
```

Output :

94.6

3. Passing a multi-dimensional array to a function :

a. The function must be called by passing only the array name.

b. In the function definition, we must indicate that the array has two dimensions by including two sets of brackets.

c. The size of the second dimension must be specified.

d. The prototype declaration should be similar to the function header.

For example :

```
#include<stdio.h>
void displayArray(int arr[2][2]);
int main()
{
    int arr[2][2], i, j;
    printf("Please enter 4 numbers for the array: \n");
    for (i = 0; i < 2; ++i)
    {
    for (j = 0; j < 2; ++j)
    {
    scanf("%d", &arr[i][j]);
    }
    }
                        // passing the array as argument
    displayArray(arr);
    return 0;
}
void displayArray(int arr[2][2])
{
    int i, j;
    printf("The complete array is: \n");
    for (i = 0; i < 2; ++i)
    {
                        // getting cursor to new line
    printf("\n");
    for (j = 0; j < 2; ++j)
    {
                        // it is used to provide tab space
    printf("%d\t", arr[i][j]);
    }
    }
}
```

Output :

```
Please enter 4 numbers for the array :
1
2
3
```

4
The complete array is:
1 2
3 4

Que 3.23. Explain parameter and its types. How can we pass parameter to function ?

OR

What do you mean by parameter passing mechanism ?

AKTU 2015-16(Sem-2), Marks 10

OR

Describe call by value and call by reference with example.

AKTU 2014-15(Sem-2), Marks 10

OR

Explain call by value and call by reference mechanism for function call using proper example.

AKTU 2016-17(Sem-2), Marks 10

OR

What do you mean by parameter passing ? Discuss various types of parameter passing mechanism in C with example.

AKTU 2016-17(Sem-1), Marks 10

Answer

1. Parameters are variable used in function definition.
2. Parameters must be written within parenthesis followed by the name of function in function definition.
3. There are two types of parameters and they are as follows :
   a. **Actual parameters :** The actual parameters are the parameters that are specified in calling function
   b. **Formal parameters :** The formal parameters are the parameters that are declared at called function.
4. When a function gets executed, the copy of actual parameter values is copied into formal parameters.

Parameter passing mechanism :

a. It is a mechanism of passing values from one function to another function.
b. There are two methods to pass parameters from calling function to called function and they are as follows :

1. Call by value :

i. When the value of arguments is passed from the calling function to the called function, the values are copied into the called function.

ii. If any changes are made to the values in the called function, then there is no change in the original values within the calling function.

For example :

```
#include<stdio.h>
main ()
{
    int n1, n2, x;
    int call_by_val();
    n1 = 6;
    n2 = 9;
    printf ("n1 = %d and n2 = %d\n", n1, n2);
    x = call_by_val(n1, n2);
    printf("n1 = %d and n2 = %d\n", n1, n2);
    printf ("x = %d", x);
}   /*end of main ()*/
/* function to illustrate call by value*/
call_by_val(p1, p2)
int p1, p2;
{
    int sum;
    sum = p1 + p2;
    p1 + = 2;
    p2 + = p1;
    printf ("p1 = %d and p2 = %d\n", p1, p2);
    return (sum);
}
```

P1 = P1 + 2
P2 = P2 + P1

Output :

```
n1 = 6      and  n2 = 9
p1 = 8      and  p2 = 17
n1 = 6      and  n2 = 9
x = 15
```

There is no change in the values of n1 and n2 before and after the function execution.

2. Call by reference :

i. In this method, the actual values are not passed, instead their addresses are passed.
ii. There is no copying of values since their memory locations are referenced.
iii. If any modification is made to the values in the called function, the original values get changed within the calling function.

For example :

```
#include<stdio.h>
main()
{
        int num[20], i, max;
```

```
        printf ( "Enter the size of the array : \n") ;
        scanf("%d", &max);
        printf ("Start entering the number \n") ;
        for(i = 0; i < max; i++)
        {
                scanf ("%d", &num[i]);
        }
                sort_nums (num, max);    /*function reference*/
                printf("Sorted numbers are as follows \n") ;
                for(i = 0; i < max; i++)
                printf("%3d", num[i]);
} /*end of main()*/
/* function to sort list of numbers*/
void sort_nums(a, n)
int a[], n;
{
        int i, j, dummy;
        for(i  = 0; i < n – 1;  i ++)
        {
                for(j = i + 1; j < n; j++)
                {
                        if(a[i] > a[j])
                        {
                                dummy = a[i];
                                a[i] = a[j];
                                a[j] = dummy;
                        }       /*End of if*/
                }       /*End of j for loop*/
        }               /*End of i for loop*/
        return;
        } /*End of function*/
```

Output :

```
Enter the size of the array :
3
Start entering the numbers :
44
64
16
Sorted numbers are as follows :
16
44
64
```

Que 3.24. Explain the difference between parameter passing mechanism call by value and call by reference. Which is more efficient and why ? AKTU 2017-18(Sem-1), Marks 07

OR

Write difference between call by value and call by reference with suitable example. AKTU 2013-14(Sem-1), Marks 05 AKTU 2015-16(Sem-1), Marks 10

Answer

S. No.	Basis	Call by value	Call by reference
1.	Description	A function which pass actual data or value to other function.	A function which do not pass actual data or value to other function.
2.	Arguments	A copy of actual argument is passed to respective formal arguments.	Reference to the location or address of actual arguments is passed to formal arguments.
3.	Value modification	Original value is not modified.	Original value is modified.
4.	Changes	Changes made inside the function are not reflected on other functions.	Any changes made in formal arguments will also reflect in actual arguments.
5.	Memory location	Actual and formal arguments will be created in different memory.	Actual and formal arguments will be created in same memory location.

Examples of call by value and call by reference : Refer Q. 3.23, Page 3–22E, Unit-3.

Call by value is more efficient because :

1. Changes made inside the function are not reflected on other function.
2. Actual and formal arguments will be created in different memory location.
3. Knowledge of pointer is not required.

Que 3.25. Differentiate between :

1. **Actual and formal arguments**
2. **Global and extern variables** AKTU 2016-17(Sem-1), Marks 10

OR

Distinguish between 'actual and formal arguments' and 'global and extern variables'. AKTU 2014-15(Sem-2), Marks 05

Answer

Difference between actual arguments and formal arguments :

S. No.	Actual arguments	Formal arguments
1.	The arguments that are passed in a function call are called actual arguments.	Formal parameters are written in the function prototype and function header of the definition.
2.	These arguments are defined in calling function.	The formal arguments are defined in function declaration.
3.	At the time of the call, each actual parameter is assigned to the corresponding formal parameter in the function definition.	Formal parameters are local variables which are assigned values from the arguments when the function is called.

4.	Example :
	void main ()
	{
	int num 1;
	display (num 1);
	}
	void display (int para 1)
	{
	_____
	_____
	}
	num 1 is actual parameter
		Example :
		void main ()
		{
		int num 1;
		display (num 1);
		}
		void display (int para 1)
		{
		_____
		_____
		}
		para 1 is formal parameter

Difference between global variable and extern variable :

S. No.	Global variables	Extern variables
1.	Global variables are defined only once in any of the program module and they can be accessed by all others.	Variables that are both alive and active throughout the entire program are known as extern variables.
2.	Memory is allocated.	No memory is allocated.

Que 3.26. **What are the different types of functions ? Write a program in C to sort list of names of students in an ascending order.**

AKTU 2015-16(Sem-1), Marks 10

AKTU 2013-14(Sem-1), Marks 10

Answer

Different types of function : Refer Q. 3.19, Page 3–16E, Unit-3.

Program to sort list of names of students in an ascending order :

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
char str[5][20], temp[20];
int i, j;
printf("Enter a strings to check.\n");
for(i = 1; i <= 5; i++)
    {
    scanf("%s",str[i]);
    }
printf("Alphabetically sorted list is :\n");
    for(i = 1; i <= 5; i++)
{
    for(j = 1; j <= 5; j++)
    {
    if(strcmp(str[j – 1], str[j]) > 0)
    {
    strcpy(temp, str[j – 1]);
    strcpy(str[j – 1], str[j]);
    strcpy(str[j], temp);
    }
    }
}
    for(i = 1; i <= 5; i++)
    {
    printf("\n%s",str[i]);
    }
    getch();
}
```

Que 3.27. **Write a program in C to find the sum of individual digits in a five digit number.**

AKTU 2013-14(Sem-1), Marks 05

Answer

```
#include<stdio.h>
#include<conio.h>
int sum(int n)
{
    if(n < 10)
    return(n);
```

```
else
return(n % 10 + sum (n / 10));
}
main()
{
        int s,n;
        printf("\nEnter any number:");
        scanf("%d",&n);
        s = sum(n);
        printf("\nSum of digits = %d",s);
        getch();
        return 0;
}
```

Output : Enter any number: 54251
Sum of digits = 17

Que 3.28. **What is recursion ?** **AKTU 2016-17(Sem-1), Marks 05**

OR

What do you mean by recursion ? Illustrate the use of recursive function with an example in C language.

AKTU 2013-14(Sem-2), Marks 05

Answer

1. Recursion is a programming technique that allows the programmer to express operations in terms of themselves.
2. Recursion function is a function that calls itself.
3. Recursive functions can be effectively used to solve problem where solution is expressed in terms of successively applying the same solution to subsets of the problem.
4. When we write recursive functions, we must have if statement somewhere to force the function to return without the recursive call being executed. Otherwise, the function will never return.
5. The factorial of a number *n* is expressed as a series of repetitive multiplications as :

   factorial of n = n(n – 1)(n – 2) ..... 1.

   For example,

   factorial of 4 = 4 × 3 × 1 = 24

   A function to evaluate factorial of *n* is as follows :

```
factorial (int n)
{
        int fact;
        if (n == 1)
            return(1);
        else
            fact = n*factorial (n – 1);
        return (fact);
}
```


Que 3.29. **What are the principles of recursion ? Explain in detail.**

AKTU 2015-16(Sem-2), Marks 05

OR

What are the main principles of recursion ?

AKTU 2014-15(Sem-2), Marks 10

Answer

Some basic principles which are used while designing algorithms with recursion are :

1. **Find the key step :**
   a. When beginning with the design of algorithms through recursion one should try to find out the Key step for the solution.
   b. Once we have executed the key step, find out whether the remainder of the problem can be solved in the same way, and modify the step, if necessary.
2. **Find a stopping rule :**
   a. The stopping rule indicates that the problem or a substantial part of it is done and the execution of the algorithm can be stopped.
3. **Outline your algorithm :**
   a. After determining the key step for the problem and finding the cases which are to be handled, the next step is to combine these two using an 'if' statement to select between them.
   b. The main program and the procedure for recursion are now to be written to carry the key step through until the stopping rule applies.
4. **Check termination :**
   a. Care should be taken to ensure that the recursion will always terminate after the finite number of steps and the stopping rule should also be satisfied.
   b. It should also handle the extreme cases correctly.
5. **Draw a recursion tree :**
   a. The key tool for the analysis of recursive algorithms is the recursion tree as it helps in determining the amount of memory that the program will require and, the total size of tree reflects the number of times the key step will be performed and the total time needed for the program.

Que 3.30. **Define recursive function. Write a program in C to generate fibonacci series (0, 1, 1, 2, 3, 5, 8, 1, 3 ...) using recursive function.** **AKTU 2013-14(Sem-1), Marks 05**

OR

What is recursion ? Write a program in C to generate the Fibonacci series. **AKTU 2016-17(Sem-1), Marks 05** **AKTU 2017-18(Sem-1), Marks 07**

Answer

Recursion and recursive function : Refer Q. 3.28, Page 3–28E, Unit-3.

Program to generate fibonacci series using recursion :

```
#include<stdio.h>
int fibo(int);
int main()
{
int num;
int result;
printf("Enter the nth number in fibonacci series: ");
scanf("%d", &num);
if (num < 0)
{
printf("Fibonacci of negative number is not possible.\n");
}
else
{
result = fibo(num);
printf("The %d number in fibonacci series is %d\n", num, result);
}
return 0;
}
int fibo(int num)
{
if (num == 0)
{
return 0;
}
else if (num == 1)
{
return 1;
}
else
{
return(fibo(num – 1) + fibo(num – 2));
}
```

Que 3.31. **What are the types of function ? Write a C program to find the factorial of a given number using recursion.**

AKTU 2014-15(Sem-1), Marks 10

Answer

Types of function : Refer Q. 3.19, Page 3–16E, Unit-3.

Program to find the factorial of a given number using recursion :

```
#include<stdio.h>
main( )
{
        int n,f;
        clrscr( );
        printf("\nEnter any number:");
        scanf("%d",&n);
        f = factorial(n);
        printf("\nfactorial = %d",f);
        getch( );
        return 0;
}
factorial(int n)
{
        if(n == 0)
        return(1);
        else
        return(n * factorial(n – 1));
}
```

Que 3.32. **If int $a = 2, b = 3, x = 0$; Find the value of $x = (++a, b += a)$.**

AKTU 2014-15(Sem-2), Marks 10

Answer

Given, $a = 2, b = 3$

$x = (++a, b += a)$

In $++a$, a will be increased by 1 because there is pre-increment of the value a.

Hence $a = 3$

In $b += a$ or $b = b + a$, the value of b will be 6 *i.e.*, $b = 3 + 3 = 6$

Hence, the resultant value of x is (3, 6).

Que 3.33. **A five digit positive integer is entered through the keyboard. Write a C function to calculate sum of digits of the 5-digit number :**

i. **Without using recursion**

ii. **Using recursion**

AKTU 2016-17(Sem-1), Marks 7.5

Answer

i. **Without using recursion**

```
#include<stdio.h>
#include<conio.h>
main()
```

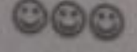
```
{
int n, d1, d2, d3, d4, d5, sum = 0;
printf("Enter the five digit number :");
scanf("%d", &n);
d1 = n%10;
n = n/10;
d2 = n%10;
n = n/10;
d3 = n%10;
n = n/10;
d4 = n%10;
n = n/10;
d5 = n;
sum = d1+ d2 + d3 + d4 + d5;
printf("The sum of digit = %d", sum);
getch();
return 0;
}
```

Output : Enter the five digit number: 12345
The sum of digit = 15

ii. Using recursion

```
#include<stdio.h>
#include<conio.h>
int sum(int n)
{
        if(n < 10)
                return(n);
        else
                return(n % 10 + sum (n / 10));
}
main()
{
        int s,n;
        printf("\nEnter any number:");
        scanf("%d",&n);
        s = sum(n);
        printf("\nSum of digits = %d", s);
        getch();
        return 0;
}
```

Output : Enter any number: 54251
Sum of digits = 17



4 UNIT

Arrays & Basic Algorithm

PART-1

Array : Array Notation and Representation, Manipulating Array Elements, Using Multi-Dimensional Array, Character Arrays and Strings.

CONCEPT OUTLINE : PART-1

- An array is a list of finite number of elements of same data type *i.e.*, integer, string etc.
- Types of array :
  i. One-dimensional array
  ii. Two-dimensional array
- String is a sequence of characters that is treated as a single data item and terminated by null character '\0'.
- String handling functions :
  i. strcat ( )
  ii. strlen ( )
  iii. strrev ( )
  iv. strcpy ( )
  v. strcmp ( )

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.1. **What is an array ? Explain about various operations of an array.**

OR

How to declare an array ? Explain about various operations of an array.

AKTU 2014-15(Sem-1), Marks 10

Answer

1. An array is a list of finite number of elements of same data type *i.e.*, integers, strings etc.
2. Syntax of array declaration is as follows :
   Data type array_name[size of array];

Operations on arrays : The operations performed on linear structure *i.e.*, arrays are :

1. **Traversal :** Processing each element in the array or list.

2. **Search :** Finding the location of the element with a given value or with a given key.
3. **Insertion :** Adding a new element to the list.
4. **Deletion :** Removing an element from the list.
5. **Sorting :** Arranging the element in some type of order, ascending or descending.
6. **Merging :** Combining two lists or arrays within a single list or array.

Que 4.2. **Explain various types of array with their declaration and initialization.**

Answer

Various types of array :

1. **One-dimensional arrays :** A one-dimensional array is a type of linear array. A list of items can be given one variable name using only one subscript and such a variable is called a one-dimensional array.

   Declaration of one-dimensional array :

   a. Arrays must be declared before they are used so that the compiler can allocate space for them in memory.

   b. The general form of array declaration is :

      type variable-name [size];

   c. The type specifies the type of element that will be contained in the array, such as int, float, or char and the size indicates the maximum number of elements that can be stored inside the array.

   d. For example : float height [50];

Initialization of one-dimensional arrays : After an array is declared, its elements must be initialized. Otherwise, they will contain "garbage value". An array can be initialized at either of the following stages :

a. **Compile time initialization :** We can initialize the elements of arrays in the same way as the ordinary variables when they are declared.

   The general form of initialization of array is :

   type array-name [size] = { list of values};

b. **Runtime initialization :** An array can be explicitly initialized at runtime. This approach is usually applied for initializing large arrays.

   For example, consider the following segment of a C program :

```
---------
---------
for (i = 0 ; i < 100; i = i + 1)
{
```

```
    if   i < 50
         sum[i] = 0.0;                    /* assignment statement */
    else
         sum[i] = 1.0;
}
---------
---------
```

The first 50 elements of the array sum are initialized to zero while the remaining 50 elements are initialized to 1.0 at runtime.

2. Two-dimensional arrays :

Declaration of two-dimensional array :

a. Two-dimensional arrays are declared as follows :

type array_name [row_size] [column_size] ;

b. Two-dimensional arrays are defined in much same manner as one dimensional array, except that a separate pair of square bracket is required for each subscript. Thus, a two-dimensional array will require two pairs of square brackets.

c. Two-dimensional arrays are stored in memory, as shown in Fig. 4.2.1.

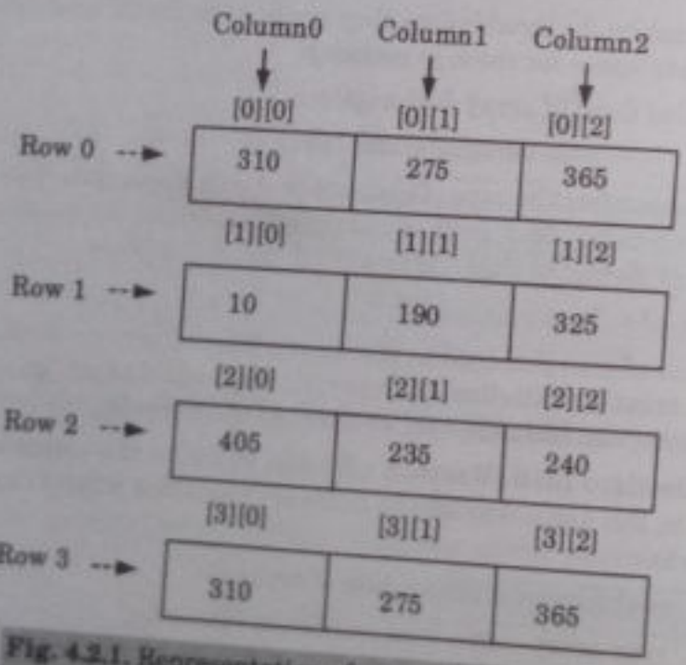


Fig. 4.2.1. Representation of a two-dimensional array in memory.

Initialization of two-dimensional arrays : Two-dimensional arrays may be initialized by following their declaration with a list of initial values enclosed in braces. For example,

int table [2] [3] = { 0, 0, 0, 1, 1, 1,} ;

initializes the elements of the first row to zero and the second row to one. The initialization is done row by row.

3. Multi-dimensional arrays : C allows arrays of three or more dimensions. The exact limit is determined by the compiler. The general form of a multi-dimensional array is :

type array_name [s1] [s2] [s3] .... [sm] ;

where s_i is the size of the i^{th} dimension. Some examples are :

int survey [3] [5] [12] ;
float table [5] [4] [5] [3] ;

Que 4.3. Write a short note on manipulating array elements.

Answer

1. Once the array is declared and defined, individual elements of array can be accessed in different ways.
2. The subscript in the square bracket indicates the position of element in the array.
3. All array elements are numbered.
4. The number starts with 0.
5. In general form, the statement

   Var_name = array_name[i];

   is used to read data from an array, where subscript i indicates the position of array elements within the array.
6. The data type of variable and array must be same.

Que 4.4. Write a short note on string and character array. How it is declared and initialized ? State some library functions for string manipulation.

Answer

1. String is a sequence of characters that is treated as a single data item and terminated by null character '\0'.
2. C language does not support strings as a data type.
3. A string is one-dimensional array of characters in C language.
4. These are often used to create meaningful and readable programs.

   For example : The string "hello world" contains 12 characters including '\0' character which is automatically added by the compiler at the end of the string.

   Declaring and initializing a character array :

   There are different ways to initialize a character array variable.

```
char name[13] = "StudyTonight";     // valid character array initialization
char name[10] = {'L','e','s','s','0','n','s','\0'};   // valid initialization
```

when we initialize a character array by listing all of its characters separately then we must supply the '\0' character explicitly.

String handling functions :

1. There are large numbers of string handling functions that can be used to carry out many of the string manipulations.
2. These functions are packaged in string.h library.
   The following are the most commonly used string handling functions.

Method	Description
strcat()	It is used to concatenate(combine) two strings
strlen()	It is used to show length of a string
strrev()	It is used to show reverse of a string
strcpy()	Copies one string into another
strcmp()	It is used to compare two string

Que 4.5. **Write a program to multiply two matrices (read size and number of element of matrices from the keyboard).**

AKTU 2017-18(Sem-1), Marks 07

OR

Write a program in C to multiply the two matrices of N × N.

AKTU 2013-14(Sem-1), Marks 05

OR

Write a C program to find the multiplication of two matrices.

AKTU 2014-15(Sem-1), Marks 10

OR

Write a program to multiply two matrices of dimensions 3 * 3 and store the result in another matrix.

AKTU 2016-17(Sem-1), Marks 05

Answer

```
#include<stdio.h>
#include<conio.h>
main( )
{
int A[N][N] ;
int B[N][N] ;
int C[N][N] ;
int i, j, k, N ;
clrscr( ) ;
printf("Enter the value of N");
scanf("%d", & N);
printf(" Enter the elements of matrix A :");
```



```
for(i = 0 ; i < N ; i ++)
{
    for(j = 0 ; j < N ; j ++)
    {
    scanf("%d", &A[i][j]) ;
    }
}
printf("Enter the elements of matrix B:") ;
for(i = 0 ; i < N ; i ++)
{
    for(j = 0 ; j < N ; j ++)
    {
    scanf("%d", &B[i][j]) ;
    }
}
for(i = 0 ; i < N ; i ++)
{
    for(j = 0; j < N ; j ++)
    {
    C[i][j] = 0;
        for(k = 0 ; k < N ; k ++)
        {
        C[i][j] = C[i][j] + A[i][k] * B[k][j] ;
        }
    }
}
for(i = 0 ; i < N ; i ++)
{
    for(j = 0 ; j < N ; j ++)
    {
    printf("\t %d", C[i][j]) ;
    }
printf("\n") ;
}
getch( ) ;
return 0 ;
}
```

Que 4.6. **Write a program in C to find the total number of words in a given sentence.**

AKTU 2013-14(Sem-2), Marks 05

Answer

```
#include<stdio.h>
#include<conio.h>
void main( )
```

```
{
    int count_words = 0,i ;
    char str[10];
    printf ("enter string\n");
    gets(str);
    for(i = 0; str[i]! = '\0'; i++)
    {
    if (str[i] = = ' ')
    count_words++;
    }
    printf("Number of words in string : %d", count_words + 1);
    getch( );
}
```

Que 4.7. **Write a program in C for construction of two matrices $A(5 \times 5)$ and $B(5 \times 5)$ of real numbers.**

AKTU 2013-14(Sem-2), Marks 05

Answer

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int A[5][5], B[5][5], row, col;
    clrscr();
    printf("Enter the elements of matrix A");
    for (row = 0; row < 5; row++)
        {
        for (col = 0; col < 5; col++)
        {
        scanf("%d", &A[row][col]);
        }
}
printf("Enter the elements of matrix B");
for (row = 0; row < 5; row++)
        {
        for (col = 0; col < 5; col++)
        {
        scanf("%d%d", &B[row][col]);
        }
}
printf("The Matrices are :\n");
        for (row = 0; row < 5; row++)
        {
        for (col = 0; col < 5; col++)
        {
```



```
        printf("%d%d\t", A[row][col]);
        }
        printf("\n");
        }
        for (row = 0; row < 5; row++)
        {
        for (col = 0; col < 5; col++)
        {
        printf("%d\t", B[row][col]);
        }
        printf("\n");
        }
        getch();
}
```

Que 4.8. **Write a function in C that returns the maximum and minimum element of a given matrix of size 4 × 4. Illustrate its use in a program.**

AKTU 2013-14(Sem-2), Marks 05

Answer

Program to illustrate the use of maximum and minimum function :

```
#include<stdio.h>
int findMax(int (*a)[10],int r,int c);
int main()
{
int a[10][10], i, j, max, min, r, c;
printf("Enter the elements in the matrix\n");
scanf("%d",&r);
printf("Enter the elements in the matrix\n");
scanf("%d",&c);
printf("Enter the elements in the matrix\n");
for(i = 1; i <= r; i++)
{
    for(j = 1; j <= c; j++)
    scanf("%d",&a[i][j]);
}
    printf("The matrix is\n");
for(i = 1; i <= r; i++)
{
    for( j = 1; j <= c; j++)
    printf("%d" ,a[i][j]);
    printf("\n");
}
    printf("\n");
```

```
    max = findMax(a,r,c);
    min = findMin(a,r,c);
    printf("The maximum and minimum elements in the matrix is
    %d\n",max,min);
    return 0;
}
// Function to find maximum element in the matrix
int findMax(int (*a)[10], int r, int c)
{
    int t, i, j;
    t = a[1][1];
    for(i = 1; i <= r; i++)
    {
    for(j = 1; j <= c; j++)
    {
        if(a[i][j] > t)
        t = a[i][j];
    }
}
    return (t);
}
//Function to find minimum element in the matrix
int findMin(int (*a)[10], int r, int c)
{
    int t, i, j;
    t = a[1][1];
    for(i = 1; i <= r; i++)
    {
    for(j = 1; j <= c; j++)
    {
        if(a[i][j] < t)
        t = a[i][j];
    }
}
    return (t);
}
```

Que 4.9. **Write a program in C that accepts a text from keyboard and display that text after removing all the blank spaces and total number of characters excluding blanks in the text.**

AKTU 2013-14(Sem-2), Marks 05

Answer

```
#include<stdio.h>
#include<conio.h>
void main( )
{
    int i, j, final, counter = 0;
    int largest = 0;
    int len;
    char sen[100];
    clrscr( );
    i = 0;
    printf("Enter any sentence of 9 or more words");
    gets(sen);
    while(sen[i]! = '\0')
    {
    len = i;
    while 'sen[i]! = " ")
    {
    counter = counter + 1;
    i = i + 1;
    }
    j = i++;
    while(sen[j] = = " ")
    {
    sen[j] = sen[j + 1];
    }
    if (largest < counter)
    {
    final = len;
    largest = counter;
    }
    i = i + 1;
    }
    printf ("Text without blank spaces is : \n");
    while (sen[final]! = " ")
    {
    printf ("%c", sen[final]);
    final = final + 1;
    }
    getch( );
}
```

Que 4.10. **Write a program in C to store the percentage marks of 50 students in an array, and then store their grades in another array. The awards of grades are as follows :**

91%-100%: S; 81%-90%: E; 71%-80%: A; 61%-70%: B; 51%-60%: C; 40%-50: D; below 40%: F.

AKTU 2013-14(Sem-2), Marks 05

Answer

```
#include<stdio.h>
#include<conio.h>
#define MAXMAR 100
void main( )
{
    int Marks[50], i;
    int per, counter = 0;
    char grade;
    for (i = 0; i < 50; i++)
    {
        if (Marks[i] < = 100 && Marks[i] > = 91)
    {
    grade = 'S';
    }
        else if (Marks[i] > = 90 && Marks[i] < = 81)
    {
    grade = 'E';
    }
        else if (Marks[i] > = 71 && Marks[i] < = 80)
    {
    grade = 'A';
    }
        else if (Marks[i] > = 61 && Marks[i] < = 70)
    {
    grade = 'B';
    }
        else if (Marks[i] > = 51 && Marks[i] < = 60)
    {
    grade = 'C';
    }
        else if (Marks[i] > = 40 && Marks[i] < = 50)
    {
    grade = 'D';
    }
    else
    {
    grade = 'F';
    counter = counter + 1;
    }
        per = (Marks[i] / MAXMAR) * 100;
    }
    getch();
```



```
}
```

Que 4.11. Write a program to add two matrices of dimension 3 × 3 and store the result in another matrix.

AKTU 2014-15(Sem-2), Marks 10

OR

Write a program to add two matrices *a* and *b* of real numbers having dimension 3 × 3 and store the result in another matrix.

AKTU 2016-17(Sem-2), Marks 10

Answer

```
#include<stdio.h>
#include<conio.h>
main()
{
    int a[3][3];
    int b[3][3];
    int x[3][3];
    int i, j;
    printf("\nEnter the elements of matrix a:");
    for(i = 0; i < 3; i++)
        for(j = 0; j < 3; j++)
        scanf("%d",&a[i][j]);
    printf("\nEnter the elements of matrix b:");
    for(i = 0; i < 3; i++)
        for( j = 0; j < 3; j++)
        scanf("%d",&b[i][j]);
    for(i = 0; i < 3; i++)
    {
        for(j = 0; j < 3; j++)
    {
        x[i][j] = a[i][j] + b[i][j];
        printf("%d",x[i][j]);
    }
        printf("\n");
    }
        getch();
        return 0;
}
```

PART-2

Structure, Union, Enumerated Data Types, Arrays of Structures, Passing Array to Functions.

CONCEPT OUTLINE : PART-2

- Structure is a user-defined data type which allows us to combine data of different types together.
- Union are similar to structure, only the difference is in terms of storage.
- Enumeration (or enum) is a user-defined datatype which is used to assign names to integral constants, the names make a program easy to read and maintain.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.12. What do you mean by structure ? How we define and declare a structure variable ?

Answer

1. Structure is a user-defined data type which allows us to combine data of different types together. Structures are used to represent a record.
2. Structure use struct keyword to hold the information of structure member or elements.
3. The general syntax of a structure definition is as follows :

```
Struct tag_name
{
    data_type     member 1 ;
    data_type     member 2 ;
    .........     .........
    .........     .........
} ;
```

4. Each member in the structure may belong to a different type of data.

Declaring structure variables :

1. The tag_name may be used subsequently to declare variables that have the tag's structure.
2. It includes the following elements :
   a. The keyword struct.
   b. The structure tag name.
   c. List of variable names separated by commas.
   d. A terminating semicolon.
3. Declaration of structure variable is as follows :


```
struct book_bank
{
    char    title[20] ;
    char    author [15] ;
    int     pages ;
    float   price ;
}
struct book_bank book1, book2, book3;
```

Here, the structure declares book1, book2 and book3 as variables of type struct book_bank. Each of these variables has four members.

Que 4.13. How to access structure members ? Discuss structure initialization.

Answer

Accessing structure members :

1. Variables are linked to the structure variables in order to make them meaningful members.
2. The link between a member and a variable is established using the member operator which is also known as 'dot operator' or 'period operator'. For example,

   book1.price

   is the variable representing the price of book1 and can be used for any other ordinary variable.

Initialization :

A structure variable can be initialized at compile time.

For example :

```
main ( )
{
    struct
    {
        int weight ;
        float height ;
    }
    student ={60, 180. 75} ;
    .....
    .....
    }
```

In the give program the value 60 is assigned to student.weight and 180.75 is assigned to student.height at compile time. There is a one-to-one corresponding between the members and their initializing values.

Que 4.14. Write difference between structure and array. Write a program in C to find the largest element of a 3 × 3 matrix.

AKTU 2013-14(Sem-1), Marks 10

Answer

Difference between structure and array :

S. No.	Structure	Array
1.	Structure is a collection of logically related data items of different data types.	An array is a collection of related data elements of same type.
2.	Structure is a user-defined data type.	An array is a derived data type.
3.	A structure needs to be first designed and declared before the variables of that type can be declared and used.	An array behaves like a built-in data type as we need to simply declare an array variable and use it.

4.	**For example :**
	struct test
	{
	int a;
	char b;
	float c;
	};
	struct test t1;
	t1.a = 99;
	t1.b = 'c';
	t1.c = 88.79;
		For example :
		int a[3] = {1, 2, 3};

Program :

```
#include<stdio.h>
#include<conio.h>
main()
{
        static int a[3][3];
        int i, j, row, col, gr;
        printf("\nEnter the elements of matrix :");
        for(i = 0; i < 3; i++)
        {
```



```
        for(j = 0; j < 3; j++)
        {
        scanf("%d",&a[i][j]);
        }
        }
        gr = a[0][0];
        row = 0;
        col = 0;
        for(i = 0; i < 3; i++)
        {   for(j = 0; j < 3; j++)
            {
              if(a[i][j] > gr)
              {
                      gr = a[i][j];
                      row = i;
                      col = j;
              }
            }
        }
        printf("Largest element is %d",gr);
        printf("\n and is situated at row =%d and col =%d",row,col);
        getch();
        return 0;
}
```

Que 4.15. Define a structure. Write a program in C to create a database of fifty students to store personal details such as roll number, name and marks. Print all the details of students whose name is entered by the user. **AKTU 2016-17(Sem-1), Marks 05**

Answer

Structure : Refer Q. 4.12, Page 4–14E, Unit-4.

Program :

```
#include<stdio.h>
struct student
{
        char name[50];
        int roll;
        float marks;
} s;
int main()
{       int i;
        printf("Enter information:\n");
        for(i = 0; j < 50; i++){
        s[i].roll = i + 1
```

```
printf("\n For roll number %d, \n", s[i].roll);
print("\n");
}
printf("Enter name: ");
scanf("%s", &s[i].name);
printf("Enter marks: ");
scanf("%f", &s[i].marks);
printf("Displaying Information:\n");
for(i = 0; i < 50; ++ i)
{
printf("Roll number: %d\n", i + 1);
printf("Name: ");
puts(s[i].name);
printf("Marks: %f\n", s[i].marks);
printf("\n");
}
return 0;
}
```

Que 4.16. **Define structure with syntax. Also write a program that compares two given dates. To store date use structure say date that contains three members namely date, month and year. If the dates are equal then display message as "Equal" otherwise "Unequal".**

AKTU 2017-18(Sem-1), Marks 07

Answer

Structure with syntax : Refer Q. 4.12, Page 4–14E, Unit-4.

Program :

```
#include<stdio.h>
#include<conio.h>
struct date
{
int day;
int month;
int year;
};
void main()
{
struct date d1,d2;
clrscr();
printf("Enter first date(dd/mm/yyyy):");
scanf("%d%d%d",&d1.day,&d1.month,&d1.year);
printf("\nEnter second date(dd/mm/yyyy):");
scanf("%d%d%d",&d2.day,&d2.month,&d2.year);
if((d1.day == d2.day)&&(d1.month == d2.month)&&
```



```
(d1.year == d2.year))
printf("\nEqual");
else
printf("\nUnequal");
getch();
}
```

Que 4.17. **Write a short note on array of structures.**

Answer

1. An array of structure is defined whenever the same structure is to be applied to a group of items, elements.
2. In array of structure each element is the structure in itself.
3. In array of structures each element of the array represents a structure variable.
4. Array of structure can be declared as :

   struct class student[100];

   defines an array called student, that consists of 100 elements. Each element is defined to be of the type struct class.
5. An array of structures is stored inside the memory in the same way as a multi-dimensional array.

For example : Consider the following declaration :

```
struct marks
{
    int subjectl;
    int subject2;
    int subject3;
};
main()
{
    struct marks student[3] =
        {{45,68,81}, {75,53,69}, {57,36,71}};
```

This declares the student as an array of three elements student[0], student[1], and student[2] and initializes their members as follow :

```
student[0].subject1 = 45;
student[0].subject2 = 68;
    ....
    ....
student[2].subject3 = 71;
```

The array student actually looks as shown in Fig. 4.17.1.

student[0].subject1	45
.subject2	68
.subject3	81
student[1].subject1	75
.subject2	53
.subject3	69
student[2].subject1	57
.subject2	36
.subject3	71

Fig. 4.17.1. The array student inside memory.

Que 4.18. Explain union with example.

Answer

1. Union is a user-defined data type that refers to a memory location using several data types.
2. Unions are conceptually similar to structures.
3. A union is declared using the union keyword.
4. In union each members uses a single shared memory location which is equal to the size of its largest data member.
5. The syntax of union declaration is as follows :

```
union name
{
data type1 member1;
      ⋮
data typeN memberN;
} variable_name1, variable_name2 .....;
```

For example :

```
union item
{
    int m;
    float x;
    char c;
}It1;
```

a. This declares a variable It1 of type union item.
b. This union contains three members each with a different data type.

Que 4.19. Define union. Write a program in C to find the record of a student having maximum marks from the list of 10 records. Each record has roll number, name, class and marks field's sentence.

AKTU 2013-14(Sem-1), Marks 10

Answer

Union : Refer Q. 4.18, Page 4–20E, Unit-4.

Program to find the record of a student having maximum marks :

```
#include<stdio.h>
#include<conio.h>
void main()
{
    struct student
    {
    char name[50],cclass[50];
    int rollno[50];
    int marks[50];
    }s[10];
    int i,pos = 0,max;
    clrscr();
    for(i = 0; i < 10; i++)
    {
    printf("Enter student name");
    scanf("%c",&s[i].name);
    printf("\nEnter student class");
    scanf("%c",&s[i].cclass);
    printf("\nEnter student rollno");
    scanf("%d",&s[i].rollno);
    printf("\nEnter student marks");
    scanf("%d",&s[i].marks);
    }
    max = s[0].marks;
    for(i = 1; i < 10; i++)
    {
    if( s[i].marks > max)
    {
```

```
    max = s[i].marks;
    pos = i;
    }
    }
    printf("Record of student having maximum marks \n");
    printf("\n%d\t%c\t%c\t%d",s[pos].rollno,s[pos].name,s[pos].class
    s[pos].marks);
    getch();
    }
```

Que 4.20. **Create a union for storing the information about various departments in a university (assume appropriate attributes to store the information). List all the departments having more than 20 faculties.**

AKTU 2013-14(Sem-2), Marks 05

Answer

```
#include<stdio.h>
#include<conio.h>
union univ
{
    char dept_name [80];
    int fac_num;
}
void main ()
{
        union univ dept [100];
        int n, i;
        printf"enter number of department : \n");
        scanf"%d", &n);
        for(i = 0; i < n; i++)
    {
        printf"enter name of department");
        gets(dept[i]. dept_name );
        printf"enter number of faculty member \n");
        scanf"%d; & dept [i]. fac_num");
    }
    printf"Name of department having faculty member more than
    20 : \ n");
    for(i = 0; i < n; i++)
    if (dept [i]. fac_num > 20)
    printf"dept[i]. dept_name \n");
    getch( );
}
```


Que 4.21. **Write a program in C to store the names of 30 students in class in an array using pointers and then display the names of those students having less than 8 characters in their names.**

AKTU 2013-14(Sem-2), Marks 05

Answer

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main( )
{
    char *student [30];
    int i, l;
    clrscr( );
    printf("Enter the names of students :\n");
    for(i = 0; i < 30; i++)
    gets(student [i]);
    printf("Names of students having less than eight characters:\n");
    for(i = 0; i < 30; i++)
        {
            l = strlen (student[i]);
            if (l < 8)
            puts(student [i]);
        }
        getch();
}
```

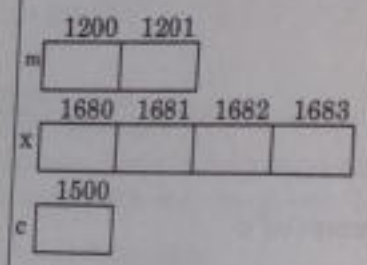
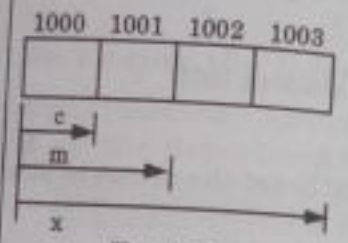
Que 4.22. **Differentiate structure and union in C. Write a C program to store the student detail using union.**

AKTU 2014-15(Sem-1), Marks 10

Answer

Difference between structure and union :

S. No.	Structure	Union
1.	A structure is a collection of data items of different data types.	It is a collection of data items of different data types.
2.	Each member has its own storage location.	All the members have the same storage location.
3.	All the members of structures can be accessed at any time.	Only one member of union can be accessed at a time.

4.	**Syntax of structure :**
	struct item
	{
	int m;
	float x;
	char c;
	};
	storage location in memory :
	 
	
	Total 7 bytes
		Syntax of union :
		union item
		{
		int m;
		float x;
		char c;
		};
		storage location in memory :
		 
		
		Total 7 bytes

5.	struct item s;
	s.m = 4;
	s.x = 386.9;
	printf("%d", s.m);
	There will be no error in printing the value of *m*.
		union item u;
		u.m = 4;
		u.x = 386.9;
		printf("%d", u.m);
		There will be error because the storage location is currently assigned to *x*. So, we cannot print the value of *m*. Hence, only one member can be accessed at a time.

Program to store the student details using union :

```
#include<stdio.h>
#include<conio.h>
void main()
{
        struct student
        {
        char name[30];
        char sex;
        int rollno;
        float percentage;
        };
        union details
        {
        struct student st;
        };
        union details set;
```



```
        printf("Enter details:");
        printf("\n Enter name : ");
        scanf("%s", &set.st.name);
        printf("\n Enter roll no : ");
        scanf("%d", &set.st.rollno);
        flushall();
        printf("\n Enter sex : ");
        scanf("%c", &set.st.sex);
        printf("\n Enter percentage :");
        scanf("%f", &set.st.percentage);
        printf("\n The student details are : \n");
        printf("\n name : %s", set.st.name);
        printf("\n Rollno : %d", set.st.rollno);
        printf("\n Sex : %c", set.st.sex);
        printf("\n Percentage : %f", set.st.percentage);
        getch();
}
```

Que 4.23. **What are enumerated data types ?**

OR

What is enumerated data type ? Write a C program to display month in the year using enum. **AKTU 2014-15(Sem-1), Marks 10**

OR

What are the enumerated data types ? Explain in detail. **AKTU 2015-16(Sem-2), Marks 10**

Answer

1. Enumeration (or enum) is a user-defined data type in C.
2. It is used to assign names to integral constants which make a program easy to read and maintain.
3. The keyword enum is used to declare new enumeration types in C.
4. Enum declaration is a follows :
   enum flag{constant1, constant2, constant3, ... };
5. Variables of type enum can be defined in two ways :
   a. enum week{Mon, Tue, Wed};
      enum week day;
   b. enum week{Mon, Tue, Wed}day;

   For example :

```
#include<stdio.h>
```

```
enum week{Mon, Tue, Wed, Thus, Fri, Sat, Sun};
int main()
{
    enum week day;
    day = Wed;
    printf("%d",day);
    return 0;
}
```

Output:

2

Program to display months in the year using enum :

```
#include<stdio.h>
#include<conio.h>
void main(void)
{
    enum month{january=1,february=2,march=3,april=4,may=5,
    june=6,july=7,august=8,september=9,october=10,
    november=11,december=12};
    printf("Here are the months name");
    printf("jan :\t%d\n",january);
    printf("feb :\t%d\n",february);
    printf("mar :\t%d\n",march);
    printf("apr :\t%d\n",april);
    printf("may :\t%d\n",may);
    printf("jun :\t%d\n",june);
    printf("jul :\t%d\n",july);
    printf("aug :\t%d\n",august);
    printf("sep :\t%d\n",september);
    printf("oct :\t%d\n",october);
    printf("nov :\t%d\n",november);
    printf("dec :\t%d\n",december);
    getch();
}
```

Que 4.24. **How to pass array to function ?**

Answer

Refer Q. 3.22, Page 3–19E, Unit-3.

PART-3

Basic Algorithm : Searching & Basic Sorting Algorithm (Bubble, Insertion and Selection), Finding Roots of Equation, Notion of Order of Complexity.

CONCEPT OUTLINE : PART-3

- Searching is the process of finding the location of given elements in the linear array.
- Two searching techniques are :
  i. Linear search (sequential)
  ii. Binary search
- Sorting is the process of arrangement of data in desired format (increasing or decreasing).
- Notation of order of complexity :
  i. Theta notation ii. Big-oh notation
  iii. Omega notation iv. Little-oh notation
  v. Little omega notation

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.25. **What do you mean by searching ? Explain.**

Answer

1. Searching is the process of finding the location of given element in the linear array.
2. The search is said to be successful if the given element is found, *i.e.*, the element does exists in the array; otherwise unsuccessful.
3. There are two searching techniques :
   a. Linear search (sequential) b. Binary search
4. The algorithm which one chooses depends on organization of the array elements.
5. If the elements are in random order, then one have to use linear search technique, and if the array elements are sorted, then it is preferable to use binary search.

Que 4.26. **Write down algorithm for linear/sequential search technique. Give its analysis.**

Answer

LINEAR(DATA, *N*, ITEM, LOC)

Here DATA is a linear array with *N* elements, and ITEM is a given item of information. This algorithm finds the location LOC of ITEM in DATA, or sets LOC : = 0 if the search is unsuccessful.

1. [Insert ITEM at the end of DATA] Set DATA[N + 1] : = ITEM
2. [Initialize counter] Set LOC : = 1
3. [Search for ITEM]
   Repeat while DATA[LOC] ≠ ITEM
   Set LOC : = LOC + 1
   [End of loop]
4. [Successful] If LOC = N + 1, then : Set LOC : = 0
5. Exit

Analysis of linear search :

1. **Best case :** The element occur at first position and the time complexity is O(1).
2. **Worst case :** The element occur at last position and the time complexity is O(n).

Que 4.27. **Write down the algorithm of binary search technique. Write down the complexity of algorithm.**

Answer

Binary search (A, n, item, loc)

Let A is an array of n number of items, item is value to be searched.

1. beg = 0
2. end = n – 1
3. mid = (beg + end) /2
4. while ((beg ≤ end) and (a [mid]! = item))
5. if (item < a[mid]) then
6. end = mid – 1
7. else beg = mid + 1
8. mid = (beg + end) / 2
9. if (beg > end) then
10. loc = – 1
11. else loc = mid
12. Exit

The complexity of binary search is $O(\log_2 n)$.

Que 4.28. **What is difference between sequential (linear) search and binary search technique ?**

Answer

S. No.	Sequential (Linear) search	Binary search
1.	No elementary condition *i.e.*, array can be sorted or unsorted.	Elementary condition i.e., array should be sorted.
2.	It takes long time to search an element.	It takes less time to search an element.
3.	Complexity is O(n).	Complexity is $O(\log_2 n)$.
4.	It searches data linearly.	It is based on divide and conquer method.

Que 4.29. **What do you mean by sorting ? Write a short note on insertion sort.**

Answer

Sorting :

1. Sorting is a process of arrangement of data in desired format (increasing or decreasing).
2. The usefulness of sorting is that we can search any desired element efficiently.

Insertion sort :

1. In insertion sort, we insert a particular value at the appropriate place in the sorted sublist, *i.e.*, during k^{th} iteration the element $a[k]$ is inserted in its proper place in the sorted sub-array $a[1], a[2], a[3] \ldots a[k-1]$.
2. This task is accomplished by comparing $a[k]$ with $a[k-1]$, $a[k-2]$, $a[k-3]$ and so on until the first element $a[j]$ such that $a[j] \leq a[k]$ is found.
3. Then each of the elements $a[k-1]$, $a[k-2]$, $a[j+1]$ are moved one position up and then element $a[k]$ is inserted in $[j+1]^{st}$ position in the array.

Insertion-Sort (A)

1. for $j \leftarrow 2$ to length[A]
2. do key $\leftarrow A[j]$
   /*Insert $A[j]$ into the sorted sequence $A[1 \ldots j-1]$.*/
3. $i \leftarrow j-1$
4. while $i > 0$ and $A[i] >$ key
5. do $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow$ key

Analysis of insertion sort :

1. Complexity in best case is $O(n)$.
2. Complexity in average case is $O(n^2)$.
3. Complexity in worst case is $O(n^2)$.

Que 4.30. Write a short note on selection sort.

Answer

1. In selection sort, we repeatedly find the next largest (or smallest) element in the array and move it to its final position in the sorted array.
2. We can do this by swapping the element at the highest index and the largest element.
3. We then reduce the effective size of the array by one element and repeat the process on the smaller sub-array.
4. The process stops when the effective size of the array becomes 1 (an array of 1 element is already sorted).

Selection-Sort (A) :

```
1. n ← length[A]
2. for j ← 1 to n-1
3.         smallest ← j
4.         for i ← j + 1 to n
5.         if A [i] < A[smallest]
6.                 then smallest ← i
7.                         exchange (A[j], A[smallest])
```

Analysis of selection sort :

1. Complexity in best case is $O(n^2)$.
2. Complexity in average case is $O(n^2)$.
3. Complexity in worst case is $O(n^2)$.

Que 4.31. Discuss bubble sort.

Answer

Bubble sort procedure is based on following idea :

1. Suppose if the array contains n elements, then $(n - 1)$ iterations are required to sort this array.
2. The set of items in the array are scanned again and again and if any two adjacent items are found to be out of order, they are reversed.
3. At the end of the first iteration, the lowest value is placed in the first position.

4. At the end of the second iteration, the next lowest value is placed in the second position and so on.
5. It is very efficient in sorting large number of data items. For n data items, this method requires $n(n-1)/2$ comparisons.

Bubble sort (A) :

```
1. for i ← 1 to length [A]
2.         for j ← length[A] down to i + 1
3.                 if A[j] < A[j – 1]
4. exchange (A[j], A[j – 1])
```

Analysis of bubble sort :

1. Complexity in best case is $O(n)$.
2. Complexity in worst case is $O(n^2)$.
3. Complexity in average case is $O(n^2)$.

Que 4.32. Write a C program to sort set of integers in ascending order by using bubble sort technique.

AKTU 2017-18(Sem-1), Marks 07

OR

Write a program in C to sort list of 10 integers in an ascending order. **AKTU 2013-14(Sem-1), Marks 05**

OR

Write a program to sort an array of integers in ascending order using bubble sort. **AKTU 2016-17(Sem-1), Marks 05**

Answer

```
#include<stdio.h>
#include<conio.h>
#define MAXSIZE 10
void main()
{
        int array[MAXSIZE];
        int i, j, num, temp;
        clrscr();
        printf("Enter the value of num \n");
        scanf("%d", &num);
        printf("Enter the elements one by one \n");
```

```
    for (i = 0; i < num; i++)
    {
    scanf("%d", &array[i]);
    }
    printf("Input array is \n");
    for (i = 0; i < num; i++)
    {
    printf("%d\n", array[i]);
    }
    /* Bubble sorting begins */
        for (i = 0; i < num; i++)
        {
        for (j = 0; j < (num – i – 1); j++)
        {
        if (array[j] > array[j + 1])
        {
        temp = array[j];
        array[j] = array[j + 1];
        array[j + 1] = temp;
        }
    }
}
    printf("Sorted array is... \n");
    for (i = 0; i < num; i++)
    {
    printf("%d\n", array[i]);
    }
    getch();
}
```

Que 4.33. **What do you mean by sorting ? Write a program in C to sort the given n positive integers. Also give the flow chart for the same.**

AKTU 2016-17(Sem-1), Marks 7.5

Answer

Sorting : Refer Q. 4.29, Page 4–29E, Unit-4.

Program : Refer Q. 4.31, Page 4–30E, Unit-4.

Assuming $n = 10$

Flow chart :

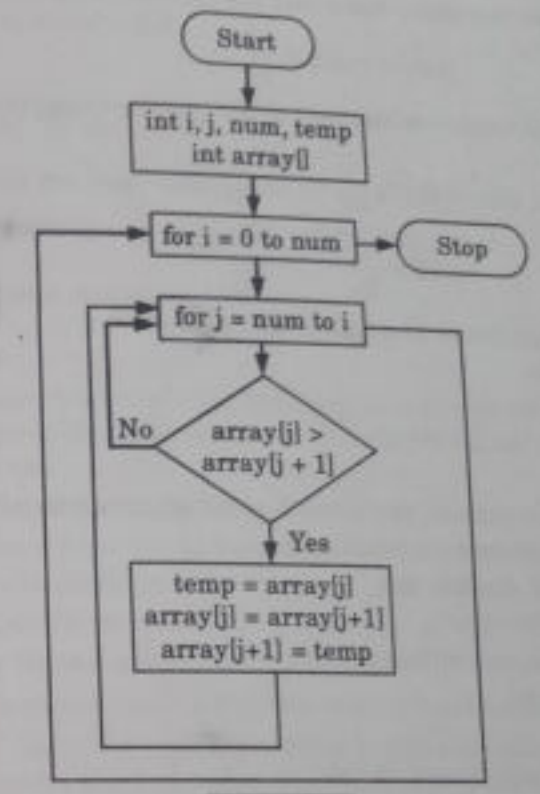


Fig. 4.33.1.

Que 4.34. **Write a short note on finding roots of quadratic equation. Write a C program to find all the roots of a quadratic equation.**

Answer

This program accepts coefficients of a quadratic equation from the user and displays the roots (both real and complex roots depending upon the determinant).

1. The standard form of a quadratic equation is :
   $$ax^2 + bx + c = 0$$
   where a, b and c are real numbers and $a \neq 0$
2. The term $b^2 - 4ac$ is known as the determinant of a quadratic equation which tells the nature of the roots.
   a. If determinant is greater than 0, the roots are real and different.

$$\text{root1} = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

$$\text{root2} = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

b. If determinant is equal to 0, the roots are real and equal.

$$root1 = root2 = \frac{-b}{2a}$$

c. If determinant is less than 0, the roots are complex and different.

$$root1 = \frac{-b}{2a} + \frac{i\sqrt{b^2 - 4ac}}{2a}$$

$$root2 = \frac{-b}{2a} - \frac{i\sqrt{b^2 - 4ac}}{2a}$$

Program to find roots of a quadratic equation :

```
#include<stdio.h>
#include<math.h>
int main()
{
double a, b, c, determinant, root1,root2, realPart, imaginaryPart;
printf("Enter coefficients a, b and c: ");
scanf("%lf %lf %lf",&a, &b, &c);
determinant = b*b - 4*a*c;
// condition for real and different roots
if (determinant > 0)
{
// sqrt() function returns square root
root1 = (-b + sqrt(determinant))/(2*a);
root2 = (-b - sqrt(determinant))/(2*a);
printf("root1 = %.2lf and root2 = %.2lf", root1 , root2);
}
//condition for real and equal roots
else if (determinant == 0)
{
root1 = root2 = -b/(2*a);
printf("root1 = root2 = %.2lf;", root1);
}
// if roots are not real
else
{
realPart = -b/(2*a);
imaginaryPart = sqrt(-determinant)/(2*a);
printf("root1 = %.2lf+%.2lfi and root2 = %.2f-%.2fi", realPart, imaginaryPart,
realPart, imaginaryPart);
}
return 0;
}
```


Output :

```
Enter coefficients a, b and c: 2.3
4
5.6
Roots are : -0.87+1.30i and -0.87 - 1.30i
```

Que 4.35. What do you understand by asymptotic notations ? Describe important types of asymptotic notations.

OR

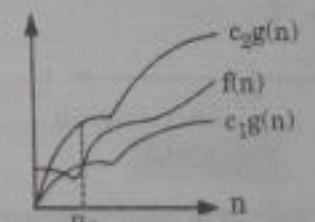
Discuss asymptotic notations in brief.

Answer

1. Asymptotic notation is a shorthand way to represent fastest possible and slowest possible running times for an algorithm, using high and low bounds on speed.
2. It is a line that stays within bounds.
3. These are also referred to as best case and worst case scenarios and are used to find complexities of functions.

Types of asymptotic notations :

1. **Θ-Notation (Same order) :**
   a. This notation bounds a function within constant factors.
   b. We say $f(n) = \Theta g(n)$ if there exist positive constants n_0, c_1 and c_2 such that to the right of n_0 the value of $f(n)$ always lies between $c_1 g(n)$ and $c_2 g(n)$ inclusive.



f(n) = Θ (g(n))

Fig. 1.35.1.

2. **O-Notation (Upper bound) :**
   a. Big-oh is formal method of expressing the upper bound of an algorithm's running time.
   b. It is the measure of the longest amount of time it could possibly take for the algorithm to complete.
   c. More formally, for non-negative functions, $f(n)$ and $g(n)$, if there exists an integer n_0 and a constant $c > 0$ such that for all integers $n \geq n_0$.

$$f(n) \leq cg(n)$$

   d. Then, $f(n)$ is big-oh of $g(n)$. This is denoted as :

$$f(n) \in O(g(n))$$

i.e., the set of functions which, as n gets large, grow faster than a constant time $f(n)$.

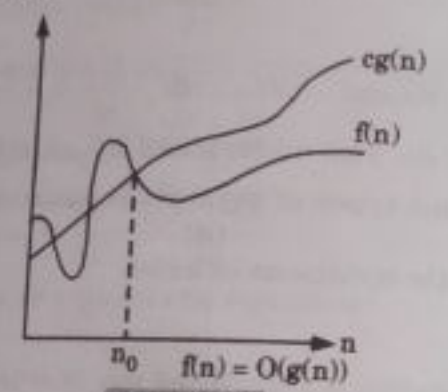


Fig. 1.35.2.

3. **Ω-Notation (Lower bound) :**
   a. This notation gives a lower bound for a function within a constant factor.
   b. We write $f(n) = \Omega g(n))$ if there are positive constants n_0 and c such that to the right of n_0, the value of $f(n)$ always lies on or above $cg(n)$.

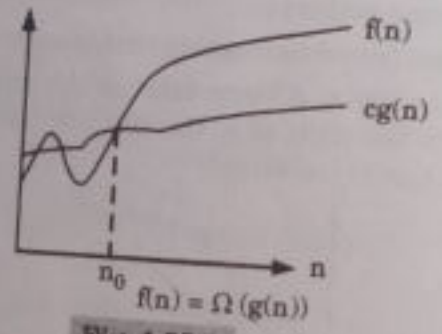


Fig. 1.35.3.

4. **Little-oh notation (o) :** It is used to denote an upper bound that is asymptotically tight because upper bound provided by O-notation is not tight.

   $o(g(n)) = \{f(n)$: for any positive constant $c > 0$, if a constant $n_0 > 0$ such that $0 \le f(n) < cg(n)\ \forall\ n \ge n_0\}$

5. **Little omega notation (ω) :** It is used to denote lower bound that is asymptotically tight.

   $\omega(g(n)) = \{f(n)$: For any positive constant $c > 0$, if a constant $n_0 > 0$ such that $0 \le cg(n) < f(n)\ \forall\ n \ge n_0\}$

☺☺☺

5 UNIT

Pointer and File Handling

PART-1

Pointers : Introduction, Declaration, Applications.

CONCEPT OUTLINE : PART-1

- **Pointers :** A variable that holds a memory address. This address is the location of another variable in memory.
- Memory to a pointer variable is assigned as : pt_name = & variable ;
  a. The & is unary operator that returns the memory address of its operand.
  b. The memory address of variable is assigned to the pointer variable pt_name.
  c. This address is the computer's internal location of the variable.
- **Advantages of pointers :**
  a. More efficient in handling arrays and data tables.
  b. Allow C to support dynamic memory management.
  c. Reduce length and complexity of programs.
  d. Efficient tool for manipulating dynamic data structures like linked lists, queues, stacks, and structures.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.1. **Write a short note on pointer.**

Answer

1. A pointer is a variable that holds the memory address of another variable (rather than data).
2. The storage and retrieval of data using the memory address of a variable (*i.e.*, a pointer) is faster than that of using variable.

Declaring pointer variables :

1. The declaration of a pointer variable takes the following form :

```
data_type *pt_name;
```

This tells the compiler three things about the variable pt_name :

a. The asterisk (*) tells that the variable pt_name is a pointer variable.
b. pt_name needs a memory location.
c. pt_name points to a variable of type data_type.

Example :

```
int *p;        /* integer pointer */
```

2. Programmers use the following styles :

```
int*   p;     /*'style 1*/
int    *p;    /* style 2 */
int  *  p;    /* style 3 */
```

Initialization of pointer variables :

1. Initialization is a process of assigning the address of a variable to a pointer variable.
2. Once a pointer variable has been declared, we can use the assignment operator to initialize the variable.

Example :

```
int   total ;
int   *p ;          /* declaration */
p =   &total ;      /* initialization */
```

3. We can also combine the initialization with the declaration. That is,

```
int *p = &total
```

Pointers are essential because of the following reasons :

a. Passing arguments to functions by address when modification of formal arguments is to be reflected on actual arguments.
b. Passing arrays and strings more conveniently from one function to another.
c. Manipulating arrays more easily by moving pointers to them instead of moving the arrays themselves.
d. Creating complex data structure, such as linked lists and binary tree, where one data structure must contain references to other data structures.
e. Communicating information about memory as in the function malloc( ) which returns the location of free memory by using a pointer.

Que 5.2. **State the features of pointers. Write a C program to sort a given number using pointers.**

AKTU 2014-15(Sem-1), 2015-16(Sem-1); Marks 10

OR

What do you mean by pointers ? How pointer variables are initialized ? Write a program to sort a given numbers using pointers.

AKTU 2016-17(Sem-1), Marks 10

Answer

Pointer and its initialization : Refer Q. 5.1, Page 5–2E, Unit-5.

Features of pointers :

1. Pointers reduce length and complexity of the program.
2. Use of pointers in our program increases the execution speed.
3. If we define an array in a function with automatic storage class, we cannot pass the address of that array back to main( ).
4. A pointer contains garbage value (any unpredictable value) until it is initialized.
5. Pointer variable can be incremented or decremented.
6. Arithmetic operation between pointer and constant is permitted but arithmetic operation between two pointers is not permitted.

Program to sort a given number using pointers :

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
void main()
{
    int n, *p, i, j, temp;
    clrscr();
    printf("\nHOW MANY NUMBER: ");
    scanf("%d", &n);
    p = (int *) malloc(n * 2);
    if(p == NULL)
    {
        printf("\nMEMORY ALLOCATION UNSUCCESSFUL");
        exit();
    }
    for(i = 0; i < n; i++)
    {
        printf("\nENTER NUMBER %d: ", i + 1);
        scanf("%d", p + i);
    }
    for(i = 0; i < n; i++)
    {
        for(j = 0; j < n; j++)
        {
            if(*(p + i) < *(p + j))
            {
                temp = *(p + i);
                *(p + i) = *(p + j);
                *(p + j) = temp;
            }
        }
    }
    printf("\nTHE SORTED NUMBERS ARE:\n");
    for(i = 0; i < n; i++)
        printf("%d ", *(p + i));
    getch();
}
```


Que 5.3. **Write a program in C to reverse a string through pointer.**

AKTU 2013-14(Sem-1), 2015-16(Sem-1); Marks 10

Answer

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
    char a[20], *ptr;
    int i, len, t, back;
    printf("Enter the string : ");
    gets(a);
    len = strlen(a);
    back = len – 1;
    printf("The entered string is : ");
    puts(a);
    if(len%2 != 0)
    {
        for(i = 0; i < (len – 1) / 2; i++)
        {
            t = a[i];
            a[i] = a[back];
            a[back] = t;
            back– –;
```

```
        }
    }
    if(len%2 == 0)
    {
        for(i = 0; i < len / 2; i++)
        {
            t = a[i];
            a[i] = a[back];
            a[back] = t;
            back--;
        }
    }
    printf("The reverse of a string is : ");
    puts(a);
    getch();
    return 0;
}
```

Que 5.4. **Write a program in C to store the names of 30 students in class in an array using pointers and then display the names of those students having less than 8 characters in their names.**

Answer

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main( )
{
char *student [30];
int i, l;
clrscr( );
printf("enter the names of students :\n");
for(i = 0; i < 30; i++)
gets(student [i]);
printf("names of students having less than eight characters:\n");
for(i = 0; i < 30; i++)
    {
```



```
        l = strlen (student[i]);
        if (l < 8)
        puts(student [i]);
    }
    getch();
}
```

Que 5.5. **What do you mean by pointers ? How pointer variables are initialized ? Also, write a note on pointer arithmetic with proper example.** **AKTU 2016-17(Sem-2), Marks 10**

Answer

Pointer and its initialization : Refer Q. 5.1, Page 5-2E, Unit-5.

Pointer arithmetic :

1. **Incrementing a pointer :**
   a. Let ptr be an integer pointer which points to the memory location 5000 and size of an integer variable is 32-bit(4 bytes).
   b. Now, when we increment pointer ptr (ptr++), it will point to memory location 5004 because it will jump to the next integer location which is 4 bytes next to the current location.
2. **Decrementing a pointer :**
   a. Decrementing a pointer will decrease its value by the number of bytes of its data type.
   b. Hence, after ptr--;
      ptr will point to 4996.
      ptr--; is equivalent to ptr-(sizeof(pointer_data_type)).
3. **Subtracting pointers :**
   a. The difference between two pointers indicates "How apart the two pointers are". It gives the total number of elements between two pointers.
   b. For example, Let size of integer is 4 bytes. If an integer pointer 'ptr1' points at memory location 10000 and integer pointer 'ptr' points at memory location 10008, the result of ptr2 - ptr1 is 2.

Que 5.6. **Write a program that accesses the array element using pointer variable.**

Answer

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int i, class[6], sum = 0;
```

```
    printf("Enter 6 numbers: \n");
for(i = 0; i < 6; ++i)
{
scanf("%d",(class + i));   // (class + i) is equivalent to &class[i]
sum += *(class + i);       // *(class + i) is equivalent to class[i]
}
printf("Sum = %d", sum);
return 0;
}
```

Que 5.7. **What are the rules for performing operations on pointer variables ?**

Answer

Following rules are applied when performing operations on pointer variables :

1. A pointer variable can be assigned the address of another variable.
2. A pointer variable can be assigned the values of another pointer variable.
3. A pointer variable can be initialized with NULL or zero value.
4. An integer value may be added or subtracted from a pointer variable.
5. Two pointer variables cannot be added.
6. A pointer variable cannot be multiplied by a constant.
7. A value cannot be assigned to an arbitrary address (*i.e.*, &x = 10; is illegal).

Que 5.8. **Explain the operations on pointer variables with example.**

Answer

Following are the operations performed on pointer variables :

Assignment :

1. A pointer variable can be assigned the address of other variable.
   ```
   int *ptr, var;
   ```
   Example :
   ```
   ptr = &var;
   ```
2. If two pointer variables are pointing to the object of the same data type, they can be assigned.
   Example :
   ```
   int *ptr1, *ptr2;
   ptr1 = ptr2;
   ```
3. A pointer variable can be assigned a NULL value (NULL = 0).
   Example :
   ```
   int *ptr;
   ```



```
ptr = Null;
```

Addition and subtraction :

1. An integer value can be added to and subtracted from a pointer variable.
   Example :
   ```
   int *ptrx;
   ptrx = (ptrx + 2);
   ptrx = (ptrx – 1);
   ```
2. One pointer variable can be subtracted from another pointer variable, if both are pointing to the elements of the same array.
   Example :
   ```
   static int x[4] = (10, 20, 30, 40);
   int *ptr1, *ptr2, *ptr3, *ptr4;
   ptr1 = &x[1];
   ptr2 = &x[3];
   ptr3 = ptr2 – ptr1;
   ```

Comparison :

If two pointer variables are pointing to the objects of the same data type, then they can be compared with one another.

Example :

```
int *ptr1, *ptr2;
    then,   (ptr1 == ptr2)
            (ptr1 != ptr2)
            (ptr1 < ptr2)
            (ptr1 > ptr2)
```

etc., can be performed.

PART-2

Introduction to Dynamic Memory Allocation (Malloc, Calloc, Realloc, Free), Use of Pointers in Self-Referential Structures, Notion of Linked List (no Implementation).

CONCEPT OUTLINE : PART-2

- The memory allocation may be classified as static and dynamic allocation.
- In dynamic allocation, the required memory size is allocated while program is getting executed.
- There are four dynamic memory allocation functions :
  a. malloc( )
  b. calloc( )
  c. realloc( )
  d. free( )
- A linked list is the chain of data items connected by pointers and each item contains a pointer to the address of next item.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.9. Write a short note on the dynamic memory allocation.

Answer

1. Dynamic memory allocation is a process of allocating memory at runtime.
2. There are four memory management functions that can be used for allocating and freeing memory during program execution.
3. They are listed in Table 5.9.1.

Table 5.9.1. Memory Allocation Functions

Function	Task
malloc( )	Allocates request size of bytes and returns a pointer to the first byte of the allocated space.
calloc( )	Allocates space for an array of elements, initializes them to zero and then returns a pointer to the memory.
free( )	Frees previously allocated space.
realloc( )	Modifies the size of previously allocated space.

Que 5.10. What are the various types of memory allocation techniques ? Give its merits and demerit.

Answer

There are two types of memory allocation techniques :

1. **Static memory allocation :** Static memory allocation refers to the process of allocating memory at compile time before the associated program is executed.

   Merits :

   a. Static memory allocation has the advantage of modularizing data within a program design in the situation where these data must be retained through the runtime of the program.

   b. A defined value is global in its scope ensuring that the same immutable value is used throughout a run for consistency.

   Demerits : If the amount of memory needed is not known at compile time then one has to make a guess.

2. **Dynamic memory allocation :** Refer Q. 5.9, Page 5–10E, Unit-5.

Merits : Memory is allocated on an as needed basis. This helps in removing the inefficiencies inherent to static memory allocation.

Demerits :

a. Dynamic memory allocation is slower than static memory allocation. This is because dynamic memory allocation happens in the heap area.

b. Dynamic memory needs to be carefully deleted after use. They are created in non-contiguous area of memory segment.

Que 5.11. Explain memory allocation process associated with a C program.

Answer

1. The conceptual view of storage of a C program in memory is shown in Fig. 5.11.1.

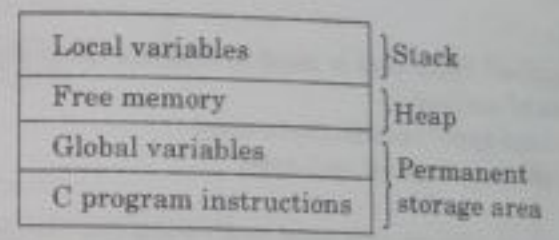


Fig. 5.11.1. Storage of a C program.

2. The program instructions, global and static variables are stored in a region known as permanent storage area and the local variables are stored in another area called stack.
3. The free memory region known as heap is located between these two regions and is available for dynamic allocation during execution of the program.
4. The size of the heap keeps changing when program is executed due to creation and death of variables that are local to functions and blocks.
5. Therefore, it is possible to encounter memory "overflow" during dynamic allocation process.
6. When memory allocation functions fail to locate enough memory requested, they return a null pointer.

Que 5.12. Explain various memory management functions used for allocation of memory with example.

OR

What do you mean by dynamic memory allocation ? Explain the following functions in detail :

i. free

ii. calloc

AKTU 2014-15(Sem-2), Marks 10

OR

What do you mean by dynamic memory allocation ? Explain the malloc( ) and calloc( ) function in detail.

AKTU 2016-17(Sem-1), Marks 10

OR

Define the concept of pointer. Also, define the dynamic memory allocation and various functions for dynamic memory allocation with suitable example.

AKTU 2017-18(Sem-1), Marks 07

Answer

Pointer : Refer Q. 5.1, Page 5–2E, Unit-5.

Dynamic memory allocation : Refer Q. 5.9, Page 5–10E, Unit-5.

Various memory management functions used for memory allocation are :

1. **malloc( ) :**
   a. The malloc( ) function is used to allocate heap storage.
   b. Syntax of malloc( ) :
      ptr = (cast-type*) malloc(byte-size)
      Here, ptr is pointer of cast-type.
2. **calloc( ) :**
   a. The calloc( ) function is used to allocate the continuous memory or element by element basis.
   b. Syntax of calloc( ) :
      ptr = (cast-type*)calloc(*n*, element-size);
      This statement will allocate contiguous space in memory for an array of *n* elements.
3. **realloc( ) :**
   a. The realloc( ) function is used to increase or decrease the size of the block of heap memory.
   b. Syntax of realloc( ) :
      ptr = realloc(ptr, newsize);
      Here, ptr is reallocated with size of newsize.
4. **free( ) :**
   a. The free( ) function is used to free a portion of storage within the heap previously allocated by alloc( ), malloc( ), calloc( ) or realloc( ).
   b. Syntax of free( ) :
      free(ptr);
      This statement frees the space allocated in the memory pointed by ptr.

Example :

Using realloc( ) and malloc( ) function :

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
```



```
    int *ptr, i , n1, n2;
    printf("Enter size of array: ");
    scanf("%d", &n1);
    ptr = (int*) malloc(n1 * sizeof(int));
    printf("Address of previously allocated memory: ");
    for(i = 0; i < n1; ++i)
    printf("%u \t",ptr + i);
    printf("\nEnter new size of array: ");
    scanf("%d", &n2);
    ptr = realloc(ptr, n2);
    for(i = 0; i < n2; ++i)
    printf("%u \t", ptr + i);
    return 0;
}
```

Using free( ) function :

```
#include<stdio.h>
#include<stdlib.h>
#define NULL 0
main()
{
char *buffer;
/* Allocating memory */
if((buffer = (char *)malloc(10)) == NULL)
{
    printf("malloc failed.\n");
    exit(1);
}
printf("Buffer of size %d created \n",_msize(buffer));
strcpy(buffer, "HYDERABAD");
printf("\nBuffer contains: %s \n", buffer);
/* Reallocation */
if((buffer = (char *)realloc(buffer, 15)) == NULL)
{
    printf("Reallocation failed. \n");
    exit(1);
}
printf("\nBuffer size modified. \n");
printf("\nBuffer still contains: %s \n",buffer);
strcpy(buffer, "SECUNDERABAD");
printf("\nBuffer now contains: %s \n",buffer);
/* Freeing memory */
free(buffer);
}
```

Que 5.13. Explain self-referential structure with its type.

Answer

Self-referential structures :

1. Self-referential structures are those structures that have one or more pointers which point to the same type of structure, as their member.
2. Structures pointing to the same type of structures are self-referential in nature.

Example :

```
struct node {
int data1;
char data2;
struct node* link;
};
int main()
{
struct node ob;
return 0;
}
```

3. In the above example 'link' is a pointer to a structure of type 'node'. Hence, the structure 'node' is a self-referential structure with 'link' as the referencing pointer.

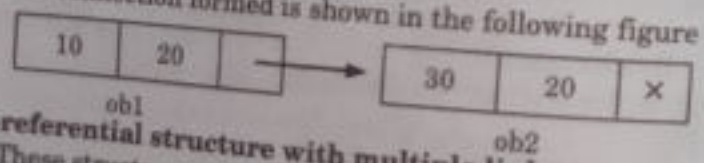
Types of self-referential structures are as follows :

1. Self-referential structure with single link :

a. These structures can have only one self-pointer as their member.

b. The following example will show us how to connect the objects of a self-referential structure with the single link and access the corresponding data members.

c. The connection formed is shown in the following figure :



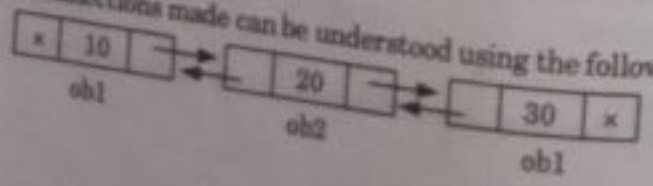
2. Self-referential structure with multiple links :

a. These structures with multiple links can have more than one self-pointer.

b. Many complicated data structures can be easily constructed using these structures.

c. Such structures can easily connect to more than one node at a time. The following example shows one such structure with more than one links.

d. The connections made can be understood using the following figure :



Que 5.14. Write a short note on data structure.

Answer

Data structure :

1. Data structure is the representation of the logical relationship existing between individual elements of data.
2. A data structure is a way of organizing all data items that considers not only the elements stored but also their relationship to each other.
3. Data structure is define as a mathematical or logical model of particular organization of data items.
4. Data structure mainly specifies the following four things :
   i. Organization of data
   ii. Accessing methods
   iii. Degree of associativity
   iv. Processing alternatives for information

The data structures are divided into following categories :

1. Linear data structure :

a. A linear data structure is a data structure whose elements form a sequence, and every element in the structure has a unique predecessor and unique successor.

b. Examples of linear data structure are arrays, linked lists, stacks and queues.

2. Non-linear data structure :

a. It is a data structure whose elements do not form a sequence. There is no unique predecessor or unique successor.

b. Examples of non-linear data structures are trees and graphs.

Que 5.15. Explain linked list in brief.

Answer

1. A linked list is a collection of records of the same data type, and all the records are connected by pointers (a variable which holds a memory address).
2. Each record in a linked list has a field that is a pointer pointing to next record in the list. The pointer field in the last record of the list is assigned the null value.
3. A pointer variable is used to point to the first record in the list.
4. The diagrammatic representation of a linked list is shown in Fig. 5.15.1

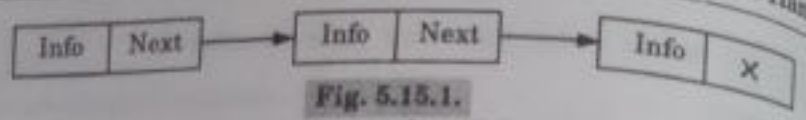


Fig. 5.15.1.

5. It is not necessary to know the number of elements and allocate memory for them beforehand. Memory can be allocated as and when necessary.
6. In case the size of the structure and the data in the structure are constantly changing then the array is not a useful structure and hence linked list is used.
7. In a linked list, insertions and deletions can be handled efficiently without having to restructure the list.

Types of linked list : There are various types of linked lists which are following :

a. **Singly linked list :** In singly linked list, each node contains data and a single link which attaches it to the next node in the list. Consider the following singly linked list representation :

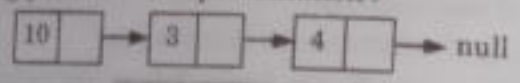


Fig. 5.15.2.

b. **Double linked list :**
   i. In double linked list, each node contains data and two links, one to the previous node and one to the next node.
   ii. Singly connected lists cannot be traversed in backward manner, with the same ease as forward traversal.
   iii. However, double linked list can be traversed forward and backward with ease as shown in Fig. 5.15.3.

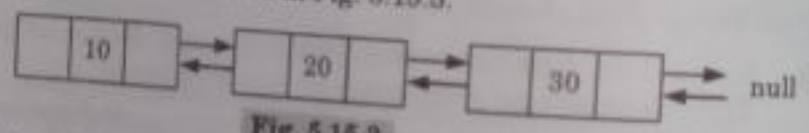


Fig. 5.15.3.

c. **Circular linked list :** In a circular linked list, the last node does not contain the NULL pointer. Instead it contains the pointer of the first node as shown in Fig. 5.15.4.

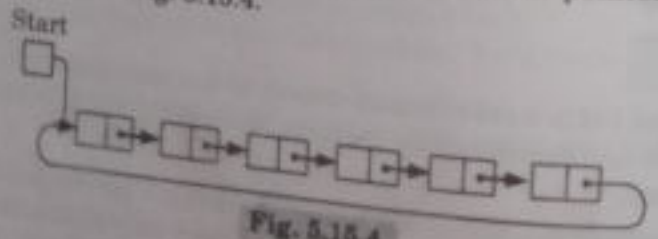


Fig. 5.15.4.

PART-3

File Handling : File I/O Functions, Standard C Preprocessor, Defining and Calling Macros, Command Line Argument.

CONCEPT OUTLINE : PART-3

- A file is a place on the disk where a group of related data is stored.
- Basic file operations :
  a. Naming a file
  b. Opening a file
  c. Reading data from a file
  d. Writing data to a file
  e. Closing a file
- The C preprocessor provides several tools that are unavailable in other high-level languages.
- The preprocessor, is a program that processes the source code before it passes through the compiler.
- Macro substitution is a process where an identifier in a program is replaced by a predefined string composed of one or more tokens.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.16. Define file. What are its types ?

Answer

1. A file is a collection of related records such as the records of employees in a company.
2. Each record will be a collection of related data items called fields. Each field consists of group of characters.
3. The common operations associated with the processing of data files are :
   a. Reading from a file (input)
   b. Writing to a file (output)
   c. Appending to a file (writing at the end)
   d. Updating a file (input or output I/O)
4. Depending upon the format in which data is stored, files are primarily categorized into two types :
   a. **Text file :**
      1. A text file stores textual information like alphabets, numbers, special symbols, etc. In actuality, the ASCII code of textual characters is stored in the text files.

2. Since data is stored in a storage device in the binary format, the text file contents are first converted in the binary form before actually being stored in the storage device.
3. A text file can store different character sets such as :
   i. Upper case English alphabets (A to Z)
   ii. Lower case English alphabets (a to z)
   iii. Numeric characters (like 1, 3, 5, etc.)
   iv. Punctuation characters (like :, ;, ',', ?, etc.)
   v. Special characters (like $, %, etc.)

b. Binary file :

1. A binary file stores the information in the binary form, *i.e.*, in the same format as it is stored in the memory.
2. Thus, the use of binary file eliminates the need of data conversion from text to binary format for storage purpose.
3. One of the main drawbacks of binary file is that the data stored in a binary file is not in human understandable form.
4. C language supports binary file operations such as read, write and append.

Que 5.17. Explain various functions of file.

Answer

Various functions of files are :

1. **getc() :**
   a. getc() function is used to read a single character from a given file and returns EOF(End-of-file) condition at the end or if an error occurs.
   b. The general format of the getc() function is : ch = getc (fptr);
   where fptr is a file pointer of the file, and ch is a variable that receives the character.
2. **fgetc() :**
   a. fgetc() is used to read a single character from a given file.
   b. The general format of the fgetc() is : ch = fgetc (fptr);
   where fptr is a file pointer of the file, and ch is a variable that receives the character.
3. **putc() :**
   a. The putc() function is used to write a single character into a file.
   b. The general format of the putc()function is : putc (ch, fptr) ;
   where ch is the character to be written and fptr is a file pointer to a file to receive a character.

4. **fputc() :**
   a. The fputc() function is used to write a single character on to a given file and advance the associated file position indicator.
   b. The general syntax of the fputc() is : fputc (ch, fptr) ;
   where ch is a character to be written and fptr is a file pointer to the file to receive the character.
5. **fputs() :**
   a. The fputs() is used to write a given string to a given file. It is normally used to copy strings from one file to another.
   b. The general format of the fputs() function is as follows : fputs (sptr, fptr)
   where sptr is a pointer to the string to be written and fptr is a file pointer to the file.
6. **fgets() :**
   a. fgets() function reads characters until it reaches a new line, an end-of-file, or the maximum number of characters specified.
   b. The format of fgets() function is : fgets (sptr, n, fptr);
7. **fread() :**
   a. The fread() function is used to read a block of binary data from a given file.
   b. The general format of the fread() function is : fread(ptr, size, nitems, fptr);
8. **fwrite() :**
   a. The fwrite() function is used to write elements from the array to the stream.
   b. The general format of fwrite() is : fwrite(ptr, size, nitems, fptr);
9. **fscanf() :**
   a. fscanf() function is used to read formatted data from a specified file.
   b. The general syntax of fscanf() function is : fscanf(fptr, "control string", &list);
10. **fprintf() :**
    a. fprintf() function is used to write formatted data into a given file.
    b. The general form of fprintf() function is : fprintf(fptr, "control string", list);
    where fptr is a file pointer to write formatted data.
11. **getw() :**
    a. getw() function is used to read an integer value from a given file.
    b. The general format of getw() is : getw(fptr);
    where fptr is a pointer to a file to receive an integer value.

12. putw() :

a. putw() function is used to write an integer quantity on to the specified file.

b. The general format of putw() function is : putw(*w*, fptr); where *w* is an integer value to be written and fptr is a file pointer to a given file.

Que 5.18. Write a C program to copy the content of one file to another.

AKTU 2013-14(Sem-1), Marks 10

Answer

```
#include<stdio.h>
#include<dos.h>
main( )
    {
        FILE *fs,*ft;
        char ch;
        clrscr( );
        if((fs = fopen("c:\\tc\\bin\\aa.cpp","r+")) == NULL)
        {
            puts("File not found.");
            exit(0);
        }
        if((ft = fopen("c:\\tc\\bin\\aaaa.txt","w")) == NULL)
        {
            puts("Target file cannot be created.");
            fclose(fs);
            exit(0);
        }
            while((ch = getc(fs)) != EOF)
        {
            putc(ch, ft);
        }
            puts("File copied successfully.");
            fclose(fs);
            fclose(ft);
    getch( );
    return 0;
    }
```

Que 5.19. What are the various types of files that can be created in C language? Also, give different modes in which these files can be used with proper syntax. Write a program in C language to append some more text at the end of an existed text file.

AKTU 2013-14(Sem-2), 2015-16(Sem-1); Marks 10

Answer

Types of file : Refer Q. 5.16, Page 5–17E, Unit-5.

Modes of accessing file :

S. No.	Mode	Meaning
1.	"r"	To open a file for reading only (for input). Reading occurs from the beginning.
2.	"w"	To open a file for writing only (for output). If the file already exists, it will be overwritten.
3.	"a"	To open a file for appending (for output). We can write only at the end-of-file position. Even if we explicitly move the file indicator, writing shall occur at the end-of-file.
4.	"r+"	To open an existing file for reading and writing (I/O). The file position indicator is initially set to the beginning of the file.
5.	"w+"	To open a new file for reading and writing (I/O). If the file already exists, its contents are destroyed. If the file does not exist, a new file is created.
6.	"a+"	To open a file for reading and appending (I/O) and create the file if it does not exist. We can read data anywhere in the file, but we can write data only at the end-of-file.
7.	"rb"	Open binary file for reading.
8.	"wb"	Create binary file for writing.
9.	"ab"	Append to a binary file.
10.	"rb+" or "r+b"	Open binary file for read/write.
11.	"wb+" or "w+b"	Create binary file for read/write.

Program :

```
#include<stdio.h>
#include<conio.h>
void main()
{
FILE *fp1,*fp2;
```

```
char ch, fname1[20], fname2[20];
printf("\n enter source file name");
gets(fname1);
printf("\n enter source file name");
gets(fname2);
fp1 = fopen(fname1, "r");
fp2 = fopen(fname2, "a");
if(fp1 == NULL || fp2 == NULL)
{
printf("unable to open");
exit(0);
}
do
{
ch = fgetc(fp1);
fputc(ch, fp2);
}
while(ch != EOF);
fcloseall();
}
```

Que 5.20. **List out various file operations in C. Write a C program to count the number of characters in file.**

AKTU 2014-15(Sem-1), 2015-16(Sem-1); Marks 10

Answer

Various file operations in C : Refer Q. 5.16, Page 5–17E, Unit-5.

Program to count the number of characters in a file :

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char ch;
    int count = 0;
    FILE *fptr;
    clrscr();
    fptr = fopen("text.txt", "w");
    if(fptr == NULL)
{
    printf"File cannot be created\a");
    getch();
    exit(0);
}
    printf"Enter some text and press enter key:\n");
    while((ch = getche()) != '\r')
{
    fputc(ch, fptr);
```



```
}
    fclose(fptr);
    fptr = fopen("text.txt","r");
    printf("\nContents of the file is:");
    while((ch = fgetc(fptr)) != EOF)
{
    count++;
    printf"%c", ch);
}
    fclose(fptr);
    printf("\nThe number of characters present in file is: %d",count);
    getch();
}
```

Que 5.21. **Suppose a file contains student records with each record containing name and age of a student. Write a C program to read these records and display them in sorted order by name.**

AKTU 2015-16(Sem-2), Marks 15

Answer

```
#include <stdio.h>
struct record
{
char name[10];
 int age;
};
void main()
{
FILE *fp,*ft;
int i,f=1;
struct record a[500];
clrscr();
fp = fopen("c:\\ctxt\\record.txt","r");
if(fp == NULL)
{
 puts("Cannot open file");
 exit();
}
for(i = 0; i < 500; i++)
{
 if(f == 1)
 {
 if((fscanf(fp, "%s%d", a[i].name, &a[i].age) == EOF))
 f = 0;
 }
 if(f == 0)
```

```
a[i].name[0] = '\0';
}
funcsort(a,4);
for(i = 0; a[i].name[0] != '\0'; i++)
printf("\nName is %s. Age is %d", a[i].name, a[i].age);
printf("\nFile sorted according to alphabetical order");
ft = fopen("c:\\ctxt\\record2.txt","w");
for(i = 0; a[i].name[0] != 0; i++)
fprintf(ft,"%s %d\n", a[i].name, a[i].age);
fclose(fp);
fclose(ft);
remove("c:\\ctxt\\record.txt");
rename("c:\\ctxt\\record2.txt","c:\\ctxt\\record.txt");
}
```

Que 5.22. **Explain five mathematical library functions in C.**

AKTU 2013-14(Sem-2), Marks 05

Answer

Following are the various mathematical library functions used in C :

S. No.	Function	Description
1.	abs ()	Returns the absolute value of an integer. **Syntax :** abs(int x);
2.	acos ()	Calculates the arc cosine. **Syntax :** double acos(double x);
3.	asin ()	Calculates the arc sine. **Syntax :** double asin(double x);
4.	atan ()	Calculates the arc tangent. **Syntax :** double atan(double x);
5.	atof ()	Converts a string to a floating point number. **Syntax :** double atof(char *s);
6.	cabs()	Absolute value of complex number. **Syntax :** double cabs(struct complex z);

Que 5.23. **What are preprocessor directive ? Explain any three of them.**

AKTU 2015-16(Sem-2), Marks 15

Answer

1. Preprocessor directives are lines included in a program that begins with the character #.
2. Preprocessor directives follow special syntax rules that are different from the normal C syntax.
3. They are invoked by the compiler to process some programs before compilation.
4. These directives can be divided into three categories :
   a. Macro substitution directives
   b. File inclusion directives
   c. Compiler control directives
5. A set of commonly used preprocessor directives and their functions is given in Table 5.23.1.

Table 5.23.1 Preprocessor Directives

Directive	Function
#define	Defines a macro substitution
#undef	Undefines a macro
#include	Specifies the files to be included
#ifdef	Test for a macro definition
#endif	Specifies the end of #if
#ifndef	Test whether a macro is not defined
#if	Test a compile time condition
#else	Specifies alternatives when #if test fails
#elif	Provides alternative test facility
#pragma	Specifies certain instructions
#error	Stops compilation when an error occurs

1. **#ifdef :** The syntax of #ifdef directive is :

   #ifdef macro_definition

Use : This directive checks whether the identifier is currently defined. Identifiers can be defined by a #define directive or on the command line.

For example :

```
#include <stdio.h>
#define RAJU 100
int main()
{
#ifdef RAJU
printf("RAJU is defined. So, this line will be added in " \"this C file\n");
#else
printf("RAJU is not defined\n");
```

```
#endif
return 0;
}
```

2. **#undef :** The syntax of #undef directive is :

```
#undef token
```

Use : The #undef directive undefines a constant or preprocessor macro defined previously using #define. Usually, #undef is used to scope a preprocessor constant into a very limited region.

For example :

```
#define E 2.71828
int e_squared = E * E;
#undef E
```

3. **#pragma :** The syntax of #pragma is :

```
#pragma compiler specific extension
```

Use : The pragma directive is used to access compiler-specific preprocessor extensions. A common use of #pragma is the #pragma once directive, which asks the compiler to include a header file only a single time, no matter how many times it has been imported.

For example :

```
#pragma once
// header file code
```

In this example, using #pragma once is equivalent to an include guard that prevents the file from being processed multiple times.

```
#ifndef _FILE_NAME_H_
#define _FILE_NAME_H_
/* code */
#endif // #ifndef _FILE_NAME_H_
```

4. **#include :** The syntax of #include directive is :

```
#include <header name>
```

Use : The include directive instructs the preprocessor to paste the text of the given file into the current file. Generally, it is necessary to tell the preprocessor where to look for header files if they are not placed in the current directory or a standard system directory.

For example :

```
#include<stdio.h>
#define PI 3.1415
#define circleArea(r) (PI*r*r)
int main()
```



```
{
int radius;
float area;
printf("Enter the radius: ");
scanf("%d", &radius);
area = circleArea(radius);
printf("Area = %.2f", area);
return 0;
}
```

Que 5.24. **Write a short note on macros with suitable example.**

AKTU 2013-14(Sem-1), Marks 10

OR

Write a short note on macros in C language.

AKTU 2013-14(Sem-2), Marks 10

OR

What do you mean by macro ? Explain types of macro with its example.

AKTU 2016-17(Sem-1), Marks 7.5

OR

What do you mean by macro ? What are the applications of macros ? Explain types of macro with its suitable examples in C language.

AKTU 2016-17(Sem-2), Marks 07

OR

What is macro ? How is it substituted ? Also explain macro act as a variable and macro act as a function with the help of example.

AKTU 2017-18(Sem-1), Marks 07

Answer

1. Macro is a preprocessor directive which is denoted by the symbol "#define".
2. C allows defining an identifier having constant value using #define directive.
3. This directive is placed at the beginning of a C program. The symbol # occurs in the first column and no semicolon is allowed at the end.
4. For example,

```
# define PI 3.14
# define MAXIMUM 100
```

Macro substitution :

1. Macro substitution is a process where an identifier in a program is replaced by a predefined string composed of one or more tokens.
2. The preprocessor accomplish this task under the direction of #define statement.
3. This statement, usually known as a macro definition (or simply a macro) takes the following general form :

   #define identifier string

4. If this statement is included in the program at the beginning, then the preprocessor replaces every occurrence of the identifier in the source code by the string.
5. There are different forms of macro substitution. The most common forms are :
   a. Simple macro substitution.
   b. Augmented macro substitution.
   c. Nested macro substitution.

Program :

```
# define PI 3.14
# include <stdio.h>
main()
{
float r = 5.25 ;
float area ;
area = PI* r * r ;
printf("\nArea of a circle = % f", areal)
}
```

Output :

Area of a circle = 86.546249

Macro as a variable :

1. A macro can be declared to accept a variable number of arguments.
2. The syntax for defining the macro is,

   #define eprintf(...) fprintf (stderr, __VA_ARGS__)

3. This kind of macro is called variadic.
4. When the macro is invoked, all the tokens in its argument list after the last named argument (this macro has none), including any commas, become the variable argument.
5. This sequence of tokens replaces the identifier __VA_ARGS__ in the macro body wherever it appears. Thus, we have this expansion :

   fprintf ("%s:%d: ", input_file, lineno)

   ⇒fprintf (stderr, "%s:%d: ", input_file, lineno)

6. The variable argument is completely macro-expanded before it is inserted into the macro expansion, just like an ordinary argument.

Macro as a function :

1. To define a function-like macro, we use the same '#define' directive, but we put a pair of parenthesis immediately after the macro name.
   For example :

   #define lang_init() c_init()
   lang_init() ⇒ c_init()

2. A function-like macro is only expanded if its name appears with a pair of parenthesis after it. If we write just the name, it is left alone.
3. This can be useful when we have a function and a macro of the same name, and we wish to use the function sometimes.

```
extern void foo(void);
#define foo() /* optimized inline version */
...
foo();
funcptr = foo;
```

   Here the call to foo()will use the macro, but the function pointer will get the address of the real function.

4. If the macro were to be expanded, it would cause a syntax error.

Application of macros :

1. The use of a macro in place of a function eliminates the time delays associated with function calls.
2. If a program contains many reported function calls, the time savings resulting from the use of macros can become significant.

Types of macros :

1. **Object-like macros :** The object-like macros is an identifier which is replaced by the value. It is widely used to represent the constant value. For example : #define mun 2.00. (Here mun is the macro name which will be replaced by value).
2. **Function-like macros :** The function macros looks like function call. For example : #define MIN(a, b) ((a) < (b) ? (a) : (b)).

Que 5.25. Explain command-line argument with example. Give its properties.

Answer

1. Command line argument is a parameter supplied to the program when it is invoked.
2. Command line argument is an important concept in C programming.
3. It is mostly used when we need to control our program from outside.
4. Command line arguments are passed to the main() method.
5. Syntax of command line argument is as follows :

   int main(int argc, char *argv[]) { /* ... */ }

   or

int main(int argc, char **argv) { /* ... */ }

Here argc counts the number of arguments on the command line and argv[] is a pointer array which holds pointers of type char which points to the arguments passed to the program.

For example :

```
#include<stdio.h>
#include<conio.h>
int main(int argc, char *argv[])
{
int i;
if( argc >= 2 )
{
printf("The arguments supplied are:\n");
for(i = 1; i < argc; i++)
{
printf("%s\t", argv[i]);
}
}
else
{
printf("argument list is empty.\n");
}
return 0;
}
```

Properties of command line arguments :

1. Command line arguments are passed to main() function.
2. Command line arguments are parameters/arguments supplied to the program when it is invoked.
3. Command line arguments are used to control program from outside instead of hard coding those values inside the code.
4. argv[argc] is a NULL pointer.
5. argv[0] holds the name of the program.
6. argv[1] points to the first command line argument and argv[n] points last argument

Que 5.26. Describe the relation between structures and pointers.

AKTU 2015-16(Sem-2), Marks 10

Answer

1. Structures can be created and accessed using pointers.
2. A pointer variable of a structure can be created as :

```
struct stock {
    char name[5];
    int item_no;
    float price;
} items[2], *ptr;
```

3. This statement items[2], *ptr; declares items as an array of two elements each of the type struct stock and pointer ptr to data objects of type struct stock.
4. The assignment statement ptr = items, would assign the address of the zeroth element of items to pointer ptr.
5. The pointer ptr will now point to items[0]. Its member can be accessed using the following statements :

```
ptr -> name
ptr -> item_no
ptr -> price
```

6. If we increment this pointer variable ptr by one, it will point to next record.
7. To print the contents of members of items, we will use printf statements like :

```
for(ptr = items; ptr < items + 2, ptr ++)
printf("%s%d%f", ptr -> name, ptr -> item_no, ptr -> price);
```


Que 5.27. What is special about void pointer ?

AKTU 2015-16(Sem-2), Marks 10

Answer

1. Void pointers are pointers that are not attached to any specific data type.
2. Such pointer variables can store addresses of any data type, float, char, integer or any other type.
3. But since they have no data type attached to it, they cannot be directly dereferenced.
4. Before accessing the memory content it should be suitably type casted to the required data type.
5. In C, we can assign a void * pointer to any other type of pointer and also any other type of pointer to a void * pointer.
6. A void * pointer is called a generic pointer. It is generally used to specify a pointer whose base type is unknown. It is also used to measure raw memory.

☺☺☺