Q.2 a. What are the steps to be followed to solve a problem using computer?

Answer:

In order to solve a problem using computer the following steps are followed:

- The given problem is analyzed.
- The solution method is broken down into a sequence of elementary tasks.
- Based on this analysis an algorithm to solve the problem is formulated. The algorithm should be precise, concise and unambiguous.
- The algorithm is then expressed in a precise notation, called computer program. The precise notation is called a computer programming language.
- The computer program is fed to the computer.
- The computer's processing unit interprets the instruction in the program, executes them and sends the results to the output unit.

b. Explain with examples, the hexadecimal representation of numbers.

Answer:

The binary equivalent of a 10-digit number will be approximately 32 bits long. Thus it is difficult to write such long strings of 1s and s and convert them to equivalent decimal numbers without making mistakes. The hexadecimal system, which uses 16 as base is a convenient notation to express binary numbers. This system by definition uses 16 symbols, viz..,0,1,2,3,4,5,6,7,8,9,A,B,.C,D,E,F. The symbols A,B etc. now represents numbers. As 16 is a power of 2, namely 2^4, there is a one to one correspondence between a hexadecimal digit and its binary equivalent. We need only 4 bits to represent a hexadecimal digit. The following table gives a hexadecimal digits and their binary and decimal equivalent.

Binary Hexadecimal and Decimal Equivalent

Decimal Number	Binary Number	Hexadecimal Equivalent
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A

11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

A binary number can be quickly converted to its hexadecimal equivalent by grouping together successively 4 bits of the binary number starting with the least significant bit and replacing each 4 bit group with its hexadecimal equivalent given in the above table.The examples below illustrate this. Examples

1. Binary number 0111 1100 1101 1110 0011

   Hexadecimal equivalent: 7 C D E 3
2. Binary number 001 0001 1111 0000 . 0010 1100

   Hexadecimal equivalent: 1 1 F 0 . 2 C

In example 2, groups are format from left to right for the fractional part of the number and from right to left for the integer part. If the number of bits in the integer part is not a multiple of 4, we insert leading 0s, as leading 0s have no significance for the integer part. If the number of bits in the fractional part is not a multiple of 4, then we introduce trailing 0s, as trailing 0s have no significance in the fractional part. Conversion from hexadecimal to decimal system is simple. It uses the fact that the base of the hexadecimal system is 16.

c. Write short notes on Error-Detecting Codes.

Answer:

Error may occur in recording data on magnetic surfaces due to bad spots on the surface. Errors may also be caused by electrical disturbances during data transmission between units. It is thus necessary to device methods to guard against such errors. The main principle used for this purpose in coded data is the introduction of extra bits in the code to aid error-detection. A common method is the use of a parity check bit along with each character code. A parity check bit is appended to the 7 bits of the code of each character in such a way that total that the total number of 1s in each character is even. For example, the ASCII code of letter E is 1000101. The number of 1s in this string is odd. A parity check bit 1 is appended to this string to obtain a code which is now 8 bits long and has an even number of 1s in it. If the ASCII code of a character has already an even number of 1s in it , then the parity check bit appended is 0. For example, the ASCII code of A is 1000001 and its code with an appended parity check bit is 10000010. All characters now have 8 bit codes including the parity check bit. Whenever a character is read from storage or received from a remote location, the number of 1s in its code is counted. It has to be even. If it is odd, then at least one bit is wrong. A single error in any of the eight bits of the code will definitely be detected. Two errors cannot be detected by this scheme as the total number of 1s in the code will remain even after two

bits change.
Instead of appending a parity check bit which makes the total number of 1s in the code even, one may choose to append a parity check bit which makes the total number of 1s in the code odd. Such a parity check bit is known as an odd parity check bit. This scheme also facilitates detection of a single error in a code.

Q.3 a. Describe the various output units of a computer.

Answer:

- **Monitor:** Monitor is an output device that resembles the television screen and uses a Cathode Ray Tube (CRT) to display information. The monitor is associated with a keyboard for manual input of characters and displays the information as it is keyed in. It also displays the program or application output. Like the television, monitors are also available in different sizes.
- **Liquid Crystal Display (LCD):** Liquid Crystal Display was introduced in the 1970s and is now applied to display terminals also. Its advantages like low energy consumption, smaller and lighter have paved its way for usage in portable computers (laptops).
- **Printer:** Printers are used to produce paper (commonly known as hardcopy) output. Based on the technology used, they can be classified as Impact or Non-impact printers. Impact printers use the typewriting printing mechanism wherein a hammer strikes the paper through a ribbon in order to produce output. Dot-matrix and Character printers fall under this category. Non-impact printers do not touch the paper while printing. They use chemical, heat or electrical signals to etch the symbols on paper. Inkjet, Deskjet, Laser, Thermal printers fall under this category of printers. When we talk about printers we refer to two basic qualities associated with printers: resolution, and speed. Print resolution is measured in terms of number of dots per inch (dpi). Print speed is measured in terms of number of characters printed in a unit of time and is represented as characters-per-second (cps), lines-per-minute (lpm), or pages-per-minute (ppm).
- **Plotter**: Plotters are used to print graphical output on paper. It interprets computer commands and makes line drawings on paper using multicolored automated pens. It is capable of producing graphs, drawings, charts, maps etc. Computer Aided Engineering (CAE) applications like CAD (Computer Aided Design) and CAM (Computer Aided Manufacturing) are typical usage areas for plotters.
- **Audio Output:** Sound Cards and Speakers: The Audio output is the ability of the computer to output sound. Two components are needed: Sound card – Plays contents of digitized recordings, Speakers – Attached to sound card.

b. What is UNIX operating system? What are the major reasons for its popularity?

Answer:

UNIX is a multiuser, time sharing operating system which was written in 1973 by Ritchie and Thomson at the Bell Telephone Laboratories, U.S.A. It was not conceived as a commercial system but was written for the convenience of a group of programmers at Bell Laboratories. It was written for a small computer – the PDP 11 manufactured by the Digital Equipment Corporation. Unlike all operating systems in the 70s, UNIX was written in a high level language – C which was a revolutionary step. By writing the operating system in C both storage and running time increased by 30% to 40%. It became a very popular operating system which has been implemented on a wide variety of computers. The variety of UNIX systems implemented use the general philosophy advocated by UNIX designers but differ in implementation details. The version of UNIX implemented by IBM on their workstations is called AIX. Hewlett Packard's version of UNIX is known as HP-UX, Sun's version is called SOLARIS and DEC's version is ULTRIX.

The major reasons for its popularity are as follows:

- UNIX is written in C, a high level language, and is thus portable to a variety of computers.
- The interface provided to users is simple but yet powerful. It provides most of the services which a user normally wants.
- The file system used by UNIX is hierarchical which allows efficient implementation and easy maintenance.
- UNIX considers all files to be a continuous sequence of characters, known as a byte stream. Application programs thus have the flexibility to format the files as per their requirements.
- UNIX treats peripheral devices as if they are files. Peripheral devices are given file names and thus programs can access devices with the same syntax they use when accessing regular files. Thus. UNIX provides a simple, consistent interface to peripheral devices.
- UNIX is multi-user, time-shared, multi-programmed operating system. Individual users can, execute several processes simultaneously. It can also be time shared by several users and this is a multitasking system.
- UNIX supports a scripting language called shell which allows complex jobs to be performed using several built-in programs provided by UNIX. The output of a shell command can be fed as input to other command, without the need to store the output in a file, by a mechanism known as a pipe.
- UNIX assumes no knowledge of the machine architecture from the user. Thus programs written using UNIX can be run on a variety of architectures.

- UNIX can support any programming language that has a compiler or an interpreter provided it has an interface that maps user requests for Operating system services to the standard set of requests used by UNIX.

Q.4 a. Differentiate between RAM and ROM.

Answer:

RAM

- It is read and write memory. User can write information into RAM and can read information from it.
- It possesses random access property, *i.e.* any memory location can be accessed in a random manner without going through any other memory location. The access search time for each memory location is same.
- It is volatile in nature *i.e.* it is volatile memory. The information written in it is retained as long as the power supply is on. As soon as the power supply goes off its stored information are lost.
- There are two types of RAM:-
  Static RAM: The static RAM consists of internal flip-flops that stores the binary information. The stored information is there as long as the power supply is on.
  Dynamic RAM: The dynamics RAM stores information in form of electric charges that are applied to capacitors. The capacitors are provided inside the RAM chip by MOS transistors. The stored charge on the capacitors are tends to discharge with time and hence the capacitors must be periodically refreshed by refreshing dynamic memory.

ROM

- ROM stands for read only memory.
- It is non-volatile *i.e.* information stored in it is not lost even if the power supply is goes-off.
- It's used for permanently storage of information.
- It possesses random-access property.
- The stored information can only be read from ROM at the time of operation and nothing can be written into ROM by user/programmer, *i.e.* it is not accessible to user.
- The contents of ROM are decided by the manufacturer and are permanently store in ROM at the time of manufacture.
- Different type of ROMs are
  (*i*) PROM : programmable ROM
  (*ii*) EPROM : erasable PROM
  (*iii*) EEPROM or E2PROM : electrically erasable PROM.

b. Briefly describe the following:
(i) Electronic Mail
(ii) World Wide Web

Answer: Refer prescribed book.

c. Define UART. What are the main functions of UART? (6

Answer: Refer prescribed book.

Q.5 a. What is a variable? What rules must be followed while constructing a variable name?

Answer:

A variable is a data name that may be used to store a data value. Unlike constants that remain unchanged during execution of a program, a variable may take different values at different times during execution. A variable name can be chosen by the programmer in a meaningful way so as reflect its function or nature in the program. Some examples of variable names are as follows:

Height, salary, counter, average etc.

Variables names may consist of letters, digits, and the underscore (_) character, subject to the following conditions:

- They must begin with a letter. Some systems permit underscore as the first character.
- ANSI standard recognizes a length of 31 characters.
- Uppercase and lowercase are significant. The variable Total and total are different.
- It should not be a keyword.
- White space is not allowed.

b. What do you mean by:
(i) Comma operator (ii) size of operator

Answer:

i) Comma operator

The comma operator can be used to link the related expressions together. A comma-linked list of expressions is evaluated left to right and the value of the right-most expression is the value of the combined expression. For example, the statement

```
value = ( x = 10, y = 5, x+y);
```

first assigns the value 10 to x, then assigns 5 to y, and finally assigns 15 to value. Since comma operator has the lowest precedence of all operators, the parentheses are necessary.

ii) sizeof operator

The sizeof operator is a compile time operator and when used with an operand, it returns the number of bytes the operand occupies. The operand may be a variable, a constant or a data type qualifier. Examples

```
m = sizeof(sum);
n = sizeof(float);
p = sizeof(235L);
```

The sizeof operator is normally used to determine the lengths of arrays and structures when their sizes are not known to the programmer. It is also used to allocated memory space dynamically to variables during execution of a program.

c. Given the values of the variable x, y and z, write a program to rotate their values such that x has the value of y, y has the value of z, and z has the value of x.

Answer:

```
#include <stdio.h>

main() {
        int x, y, z, temp;

        printf("\nEnter the value of x, y, z ");
        scanf("%d %d %d", &x, &y, &z);

        temp = x;
        x = y;
        y = z;
        z = temp;

        printf("\nValue of x after rotation is : %d", x);
        printf("\nValue of y after rotation is : %d", y);
        printf("\nValue of z after rotation is : %d", z);
}
```

Q.6 a. Explain the following with examples:
(i) if ... else statement
(ii) The ? : operator
(iii) Break statement

Answer:

(i) if ..... else statement

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)
}
next statetement;
.............
```

If the test expression is true, then true-block statements(s), immediately following the if statements are executed; otherwise, the false-block statement(s) are executed. In either case, either true-block or false-block will be executed, not both. In both cases the control is transferred subsequently to the next statement.

For example, the following program statement will find the larger of two numbers:

```
.................
if (num1 > num2) {
        printf("Number %d is larger", num1);
}
else {
        printf("Number %d is larger", num2);
.............
```

(ii) The ? : operator

The C language has an unusual operator, useful for making two-way decision. This operator is a combination of ? and :, and takes three operands. This operator is popularly known as the conditional operator. The general form of use of the conditional is as follows:

(conditional expression) ? expression1 : expression2

The conditional expression is evaluated first. If the result is nonzero, expression1 is evaluated and is returned as the value of the conditional expression. Otherwise expression2 is evaluated and its value is returned.

For example, the segment

```
if (x < 0)
        flag = 0;
else
        flag = 1;
```

(iii) break statement

An early exit from a loop can be accomplished by using the break statement. When a break statement is encountered inside the loop, the loop is immediately exited and the program continues with the statement immediately following the loop. When the loops are nested, the break would only exit from the loop containing it, i.e. the break will exit only a single loop.

The use of the break statement in loops is illustrated as follows:

(a)
```
while (test-condition) {
        .....................
        if (condition)
                break;
        ..............
        ..............
}
```

```
        next-statement;

(b)     do {
                ....................
                if (condition)
                        break;
                ...................
                ...................
        } while (test-condition);
        next-statement;


(c)     for (initialization; test-condition; update) {
                ....................
                if (condition)
                        break;
                ...................
                ...................
        }
        next-statement;
```

In all the above three situations if the condition in if-statement is found true break statement will executes and statements following continue statement will be bypassed and the control will come out of the loop and next statement following the loop will executes.

b. Write a program using while loop to find the sum and reverse the digits of a number where the number is user input. For example, if the number entered is "1234" the sum would be (1+2+3+4) and reverse of the number should be written as 4321.

Answer:

```
#include <stdio.h>

main() {

        int num, rev, sum, dig;

        printf("\nEnter any number : ");
        scanf("%d",&num);

        rev=0;
        sum=0;
```

```
        while (num != 0) {
                dig = num % 10;
                sum = sum + dig;
                rev = rev * 10 + dig;
                num = num / 10;
        }

        printf("Sum of digits is : %d", sum);
        printf("\nReverse of a Number is : %d", rev);
}
```

Q.7 a. Discuss the two methods of initialization of single-dimensional array. Give example.

Answer:

After an array is declared, its element must be initialized. Otherwise they will contain garbage. An array can be initialized at either of the following stages:

- At compile time
- At run time

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 arrays is :

```
type array_name[size] = {list of values};
```

The values in the list are separated by commas. For example the statement

```
int number[3] = {0, 0, 0);
```

will declare the variable number as an array of size 3 and will assign zero to each element. If the number of values in the list is less than the number of elements, then only that many elements will be initialized. The remaining elements will be set to zero automatically. For example:

```
float total[5] = {2.3, 4.3, 6.79};
```

will initialized the first three elements to 2.3, 4.3, 6.79 respectively and the remaining two elements to zero.

The size may be omitted. In such cases, the compiler allocates enough space for all initialized elements. For example, the statement

```
int xyz[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
```

will declare the xyz array to contain nine elements with initial values 1, 2, ... ,9. This approach works fine as long as we initialize every element in the array.

Also, if we have more initializers than the declared size, the compiler will produce an error. That is the statement

```
int array[3] = {10, 20, 30, 40, 50};
```

will not work. It is illegal in C.

Run Time Initialization

An array can be explicitly initialized at run time. This approach is usually applied for initializing large arrays. For example, consider the following segment of C program:

```
.....
.....

for (i=0; i<100; i++) {
        if (i < 50)
                sum[i] = 0.0;
        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 run time.

b. s1, s2, and s3 are three string variables. Write a program to read two string constants into s1 and s2 and compare whether they are equal or not. If they are not, join them together. Then copy the contents of s1 to the variable s3. At the end, the program should print the contents of all three variables and there lengths.

Answer:

```
#include <stdio.h>
#include <string.h>
main() {
        char s1[20], s2[20], s3[20];
        int x, l1, l2, l3;

        printf("\nEnter any two string : ");
        scanf("%s %s", s1, s2);

        x = strcmp(s1, s2);

        if(x != 0) {
                printf("\n\n Strings are not equal\n");
                strcat(s1,s2);
        }
        else
```

```
                printf("\n\nStrings are equal\n");


        strcpy(s3, s1);

        l1 = strlen(s1);
        l2 = strlen(s2);
        l3 = strlen(s3);

        printf("\nS1 = %s \t length = %d characters \n", s1, l1);
        printf("\nS2 = %s \t length = %d characters \n", s2, l2);
        printf("\nS3 = %s \t length = %d characters \n", s3, l3);
}
```

Q.8 a. What is function declaration? What are the places in a program where a function declaration is made? Is prototype declaration essential? Give reason.

Answer:

Like variables, all functions in a C program must be declared, before they are invoked. A function declaration, also known as function prototype, consists of four parts as:

- function type (return type)
- function name
- parameter list
- terminating semicolon

The general syntax for function declaration is as follows:

```
function_type function_name(parameter_list);
```

This is very similar to the function header line except the terminating semicolon. For example,

```
int mul(int m, int n);          /* FUNCTION PROTOTYPE */
```

Following points are to be noted:

- The parameter list must be separated by commas.
- The parameter names do not need to be the same in the prototype declaration and the function definition.
- The types must match the types of parameters in the function definition, in number and order.
- Use of parameter names in the declaration is optional.
- If the function has no formal parameters, the parameter list is written as (void);
- The return type is optional, when the function returns int type data.
- The return type must be void if no value is returned.
- When the declared types do not match with the types in the function definition, compiler will produce an error.

Equally acceptable forms of declaration of mul function are:

- int mul(int , int);
- mul(int m, int n);
- mul(int , int);

A prototype declaration may be placed in two places in a program:

1. Above all functions (including main);
2. Inside a function definition.

When we place the declaration above all functions (in the global declaration), the prototype is referred to as a global prototype. Such declarations are available for all the functions in the program.
When we place it in a function definition (in the local declaration), the prototype is called a local prototype. Such declarations are primarily used by the functions containing them.

The place of declaration of a function defines a region in a program in which the function may be used by other functions. This region is known as the scope of the function. It is good programming style to declare prototypes in the global declaration section before main. It adds flexibility, provides an excellent quick references to the functions used in the program, and enhances documentation.

Prototype declarations are not essential. If a function has not been declared before it is used, C will assume that its details available at the time of the linking. Since the prototype is not available, C will assume that the return type is an integer and that types of parameters match the formal definitions. If these assumptions are wrong, the linker will fail and we will have to modify the program. The moral is that we must always include prototype declarations, preferably in global declaration section.

b. Write a function "power" that computes x raised to the power y for integers x and y and returns double-type value.

Answer:

```
#include <stdio.h>

double power(int x, int y);
main() {
        int x, y;
        float result;

        printf("Enter the value of x and y ");
        scanf("%d %d", &x, &y);
        result = power(x, y);
```

```
        printf("\n %d raise to power %d is %f", x, y, result);
}

double power(int x, int y) {
        double p;
        p = 1.0;

        while (y > 0) {
                p = p * x;
                y--;
        }

        return(p);
}
```

Q.9 a. Explain, how we declare a pointer variable? What information it pass on to the compiler?

Answer:

In C, every variable must be declared for its type. Since pointer variables contain addresses that belong to a separate data type, they must be declared as pointers before we use them. 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:

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

for example, int *ptr;
declares the variable ptr as a pointer variable that points to an integer data type. Thus, the type int refers to the data type of the variable being pointed to by ptr and not the type of the value of the pointer.
Similarly, the statement float *x; declares x as a pointer to a floating-point variable.

b. Write a program using pointers to compute the sum of all elements stored in an array.

Answer:

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

```
main() {
        int *p, sum, i;

        int arr[5] = {5, 6, 7, 8, 9};

        i=0;
        p = arr;
        sum = 0;
        while ( i < 5) {
                sum     = sum + *p;
                i++;
                p++;
        }

        printf("Required Sum is %d", sum);
}
```

c. When a program is terminated, all files used by it are automatically closed. Why is it then necessary to close a file during execution of the program?

Answer:

A file must be closed as soon as all operations on it have been completed. This ensures that all outstanding information associated with the file is flushed out from the buffers and all links to the file are broken. It also prevents any accidental misuse of the file. In case, there is a limit to the number files that can be kept open simultaneously, closing of unwanted files might help open the required files. Another instance where we have to close a file is when we want to reopen the same file in a different mode. The I/O library supports a function to do this for us. It takes the following forms:

```
fclose(file_pointer)
```

This would close the file associated with the FILE pointer file_pointer. Consider the following segment of a program

```
.....
.....
FILE *p1, *p2;
p1 = fopen(“INPUT”, “w”);
p2 = fopen(“OUTPUT”, “r”);
.....
.....
fclose(p1);
fclose(p2);
.....
.....
```

This program opens two files and closes them after all operations on them are completed. Once a file is closed, its file pointer can be reused for another file. Although, all files are closed automatically whenever a program terminates. But, closing a file as soon as you are done with it is a good programming habit.

TEXT BOOKS

I. Fundamentals of Computers, V. Rajaraman, Fifth Edition, PHI, 2011
II. Programming in ANSI C, E. Balagurusamy, 5th Edition, Tata McGraw Hill, 2011