# B.Tech.

## SECOND SEMESTER EXAMINATION, 2009-10

## COMPUTER CONCEPTS AND PROGRAMMING IN C

(ECS-201)

*Time : 3 Hours]* *[Total Marks : 100*

### Section-A

This question contains 10 questions of multiple choice. Attempt all parts of this section

**Q. 1. (a) C is a :**

**(i) High Level Language**

**(ii) Low Level Language**

**(iii) High Level Language with some low level features.**

**(iv) Low Level Language with high level features.**

**Ans.** High Level Language with some low level features.

**(b) Windows Xp is a :**

**(ii) Multi user multi tasking OS** **(ii) Multi user single tasking**

**(iii) Single user multi tasking** **(iv) Single user single tasking**

**Ans.** Multi user multi tasking OS

**(c) The purpose of the following program fragment :**

```
b = s + b;
s = b- s;
b = b - s;
```

**where s, b are two integers is to :**

**(i) Transfer the content of s to b** **(ii) Transfer the content of b to s**

**(iii) Negate the content of s and b** **(iv) Swap the contents of s and b**

**Ans.** Swap the contents of s and b

**(d) Consider the function**

```
Find(int x, int y)
{return ((x < y ? 0: (x - y));}
```

**let a, b be two non negative integers. The call find (a, find(a,b)) can be used to find the:**

**(i) Maximum of a, b** **(ii) Positive difference of a, b**

**(iii) Sum of a, b** **(iv) Minimum of a, b**

**Ans.** Minimum of a, b

**(e) The maximum value of a signed integer is :**

**(i)** $2^{16} - 1$ **(ii)** $2^{15} - 1$ **(iii)** $2^{16}$ **(iv)** $2^{15}$

www.btechsamplepapers.in

**Ans.** $2^{16} - 1$

**(f) The value of an automatic variable that is declared but not initialized will be :**

**(i) 0 (ii) Unpredictable (iii) – 1 (iv) None of these**

**Ans.** Unpredictable

**(g) If n = 3 then the output of the statement printf("%d %d", n++, ++n); will be :**

**(i) 3 5 (ii) 4 5 (iii) 4 4 (iv) is implementation dependent**

**Ans.** 4 4

**(h) The following program fragment**

```
for (i = 3; i< 15; i = i +3);
printf(" %d" ,i);
```

**results in**

**(i) A syntax error (ii) An execution error**

**(iii) Printing of 12 .(iv) Printing of 15**

**Ans.** Printing of 15

**(i) 2s complement of $(5)_{10}$ will be :**

**(i) 5 (ii) 6 (iii) – 5 (iv) Not possible**

**Ans.** – 5

**(j) Select the odd man out:**

**(i) Integer (ii) Structure (iii) Union (iv) Array**

**Ans.** Integer

## Section-B

**Note** : *Attempt any* ***three*** *questions. All questions carry* ***equal*** *marks :*

**Q. 2. (a) (i) Differentiate between DOS, UNIX and Windows Operation System.**

**(ii) Explain tbe following UNIX command,**

**Is, cbmod, sh, au, who.**

**Ans.**

1. DOS uses CLI (command line interface), whereas Windows used GUI (graphical user interface)..
2. DOS does not aupport networking, Windows does..
3. DOS is a single user OS, Windows is Multiuser..
4. DOS is a single tasking OS, Windows is Multitasking..
5. Dos is a single threading OS, Windows is a Multithreading..
6. DOS supports 2 GB of maximum partition size, Windows supports 2 TB or more..
7. DOS uses FAT 16 file system, Windows uses FAT 32..
8. Server administration is not possible in DOS..

Unix and Windows use completely different paradigms for run-time loading of code. Before you try to build a module that can be dynamically loaded be aware of how your system works.

In Unix a shared object (.so) file contains code to be used by the program and also the names of functions and data that it expects to find in the program. When the file is joined to the program all references to those functions and data in the file's code are changed to point to the

www.btechsamplepapers.in

actual locations in the program where the functions and data are placed in memory. This is basically a link operation.

In Windows a dynamic-link library (.dll) file has no dangling references. Instead an access to functions or data goes through a lookup table. So the DLL code does not have to be fixed up at runtime to refer to the program's memory; instead the code already uses the DLL's lookup table and the lookup table is modified at runtime to point to the functions and data.

In Unix thera is only one type of library file (.a) which contains code from several object files (.o). During the link step to create a shared object file (.so) the linker may find that it doesn't know where an identifier is defined. The linker will look for it in the object files in the libraries; if it finds it it will include all the code from that object file.

In Windows there are two types of library a static library and an import library (both called .lib). A static library is like a Unix .a file; it contains code to be included as necessary. An import library is basically used only to reassure the linker that a certain identifier is legal and will be present in the program when the DLL is loaded. So the linker uses the information from the import library to build the lookup table for using identifiers that are not included in the DLL. When an application or a DLL is linked an import library may be generated which will need to be used for all future DLLs that depend on the symbols in the application or DLL.

**(II) ls** --- lists your files

**ls -l** --- lists your files in 'long format', which contains lots of useful information, e.g. the exact size of the file, who owns the file and who has the right to look at it, and when it was last modified.

**ls -a** --- lists all files, including the ones whose filenames begin in a dot, which you do not always want to see.

- **chmod options filename** --- lets you change the read, write, and execute permissions on your files. The default is that only you can look at them and change them, but you may sometimes want to change these permissions. For example, **chmod o+r filename** will make the file readable for everyone, and **cbmod o-r filename** will make it unreadable for others again. Note that for someone to be able to actually look at the file the directories it is in need to be at least executable. See help protection for more details.
- **who** --- tells you who's logged on, and where they're coming from. Useful if you're looking for someone who's actually physically in the same building as you, or in some other particular location.

**Q. 2. (b) Explain the various data types in C giving suitable examples of each.**

**Ans.** In C language, it is compulsory to declare variables with their data type before using them in any statement. Mainly data types are categorized into 3 categories:-

**1. Fundamental Data Types**

**2. Derived Data Types**

**3. User Defined Data Types**

Every variable used in the program should be declared to the compiler. The declaration does two things.

1. Tells the compiler the variables name.

2. Specifies what type of data the variable will hold.

The general format of any declaration

datatype v1, v2, v3, ..... vn;

www.btechsamplepapers.in

Where v1, v2, v3 are variable names. Variables are separated by commas. A declaration statement must end with a semicolon.

**Example:**

Int sum;

Int number, salary;

Double average, mean;

| Datatype | Keyword Equivalent |
|---|---|
| Character | char |
| Unsigned Character | unsigned char |
| Signed Character | signed char |
| Signed Integer | signed int (or) int |
| Signed Short Integer | signed short int (or) short int (or) short |
| Signed Long Integer | signed long int (or) long int (or) long |
| Un Signed Integer | unsigned int (or) unsigned |
| Un Signed Short Integer | unsigned short int (or) unsigned short |
| Un Signed Long Integer | unsigned long int (or) unsigned long |
| Floating Point | float |
| Double Precision Floating Point | double |
| Extended Double Precision Floating Point | long double |

www.btechsamplepapers.in

**User defined type declaration**

In C language a user can define an identifier that represents an existing data type. The user defined datatype identifier can later be used to declare variables. The general syntax is

typedef type identifier;

here type represents existing data type and 'identifier' refers to the 'row' name given to the data type.

**Example :**

typedef int salary;

typedef float average;

Here salary symbolizes int and average symbolizes float. They can be later used to declare variables as follows:

Units dept1, dept2;

Average section1, section2;

Therefore dept1 and dept2 are indirectly declared as integer datatype and section1 and section2 are indirectly float data type.

The second type of user defined datatype is enumerated data type which is defined as follows.

Enum identifier {value1, value2 .... Value n};

The identifier is a user defined enumerated datatype which can be used to declare variables that have one of the values enclosed within the braces. After the definition we can declare variables to be of this 'new' type as below.

enum identifier V1, V2, V3, ..... Vn

The enumerated variables V1, V2, ..... Vn can have only one of the values value1, value2 ..... value n

**Example :**

```
enum day {Monday, Tuesday, .... Sunday};
enum day week_st, week end;
week_st = Monday;
week_end = Friday;
if(wk_st == Tuesday)
week_en = Saturday;
```

**Q. 2. (c) Convert the following numbers as specified**

**(i)** $(1011011101.01101)_2 = (?)_{10}$ **(ii)** $(3142.28)_{10} = (?)_2$

**(iii)** $(110111101010.01101)_2 = (?)_{18}$ **(iv) 2s complement of 110100100**

**(v)** $(AC01.2DB)_2 = (?)_8$

**Ans.**

(I) $(1011011101.01101)_2 = (733.406)_{10}$

(II) $(3142.28)_{10} = (110001000110.01001)_2$

(III) $(110111101010.01101)_2 = (DEA.68)_{16}$

IV) (2S COMPLEMENT OF 110100100) = 001011100

(V) $(AC01.2DB)_2 = (1261.1333)_8$

**Q. 2. (d) Write a program in C to copy the content of a given file say "a.txt" to another file "b.txt".**

**Ans.**

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
 FILE *from, *to;
 char ch;
 if (argc!=3) {
  printf("Usage: copy <source> <destination>\n");
  exit(1);
 }
 /* open source file */
 if ((from = fopen(argv[1], "rb"))==NULL) {
  printf("Cannot open source file.\n");
  exit(1);
 }
 /* open destination file */
```

www.btechsamplepapers.in

```
if ((to = fopen(argv[2], "wb"))==NULL) {
  printf("Cannot open destination file.\n");
  exit(1);
}
/* copy the file */
while (!feof(from)) {
  ch = fgetc(from);
  if (ferror(from)) {
    printf("Error reading source file.\n");
    exit(1);
  }
  if (!feof(from)) fputc(ch, to);
  if (ferror(to)) {
    printf("Error writing destination file.\n");
    exit(1);
  }
}
if (fclose(from)==EOF) {
  printf("Error closing source file.\n");
  exit(1);
}
if(fclose(to)==EOF) {
  printf("Error closing destination file.\n");
  exit(1);
}
return 0;
}
```

**Q. 2. (e) Differentiate between call by value and call by reference. Make a program in C to show the usage of both.**

**Ans.** The arguments passed to function can be of two types namely

1. Values passed
2. Address passed

The first type refers to call by value and the second type refers to call by reference.

For instance consider program1

```
main()
{
int x=50, y=70;
interchange(x,y);
printf("x=%d y=%d",x,y);
}
interchange(x1,y1)
```

www.btechsamplepapers.in

```
int x1,y1;
{
int z1;
z1=x1;
x1=y1;
y1=z1;
printf("x1=%d y1=%d",x1,y1);
}
```

Here the value to function interchange is passed by value.

Consider **program2**

```
main()
{
int x=50, y=70;
interchange(&x,&y);
printf("x=%d y=%d",x,y);
}

interchange(x1,y1)
int *x1,*y1;
{
int z1;
z1=*x1;
*x1=*y1;
*y1=z1;
printf("*x=%d *y=%d",x1,y1);
}
```

Here the function is called by reference. In other words address is passed by using symbol & and the value is accessed by using symbol *.

The main difference between them can be seen by analyzing the output of program1 and program2.

The output of program1 that is call by value is

x1=70 y1=50

x=50 y=70

But the output of program2 that is call by reference is

*x=70 *y=50

x=70 y=50

This is because in case of call by value the value is passed to function named as interchange and there the value got interchanged and got printed as

x1=70 y1=50

and again since no values are returned back and therefore original values of x and y as in main function namely

x=50 y=70 got printed.

www.btechsamplepapers.in

## Section-C

**Note : Attempt any two parts from each question. All questions are compulsory :**

**Q. 3. (a) Write a program in C to find whether the given number is prime or not.**

**Ans.**

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

**Q. 3. (b) Write a program in C to find out second largest element of a given list of integers.**

**Ans.**

```
while ( counter <= 10 ) // Begin 1st while loop
{
  Printf("Please enter a number from the keyboard.\n" ); // Prompt user to enter a number from the keyboard
  Scanf("%d",& num); // Read in number entered by user
  Printf("\n"); // Insert blank line
  if ( num > largest ) // Begin loop to determine the largest of 10 numbers entered by user from the keyboard
       {
 seclargest = largest;
 largest = num;
       }
  else if ( num > seclargest )
    seclargest = num;
 counter++;
}
```

**Q. 3. (c) Write a program in C to sort the given list of names.**

**Ans.** Bubble sort [string array]

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 50
#define N 2000
void sort_words(char *x[], int y);
void swap(char **, char **);
int main(void) {
```

www.btechsamplepapers.in

```
char word[MAX];
char *x[N];
int n = 0;
int i = 0;
for(i = 0; scanf("%s", word) == 1; ++i) {
 if(i >= N)
  printf("Limit reached: %d\n", N), exit(1);
 x[i] = calloc(strlen(word)+1, sizeof(char));
 strcpy(x[i], word);
}
n = i;
sort_words(x, n);
 for(i = 0; i < n; ++i)
  printf("%s\n", x[i]);
 return(0);
}
void sort_words(char *x[], int y) {
 int i = 0;
 int j = 0;
 for(i = 0; i < y; ++i)
  for(j = i + 1; j < y; ++j)
   if(strcmp(x[i], x[j]) > 0)
    swap(&x[i], &x[j]);
}
void swap(char **p, char **q) {
 char *tmp;
 tmp = *p;
 *p = *q;
 *q = tmp;
}
```

**Q. 4. (a)Define the storage class in C.**

**Ans.** A storage class defines the scope (visibility) and life time of variables and/or functions within a C Program.

There are following storage classes which can be used in a C Program

auto

register

static

extern

auto - Storage Class

auto is the default storage class for all local variables.

```
{
int Count;
auto int Month;
}
```

The example above defines two variables with the same storage class. auto can only be used within functions, i.e. local variables.

register - Storage Class

www.btechsamplepapers.in

register is used to define local variables that should be stored in a register instead of RAM. This means that the variable has a maximum size equal to the register size (usually one word) and cant have the unary '&' operator applied to it (as it does not have a memory location).

```
{
registar int Miles;
}
```

Register should only be used for variables that require quick access - such as counters. It should also be noted that dafining 'register' goes not mean that the variabla will be stored in a register. It means that it MIGHT be stored in a register - depending on hardware and implimentation restrictions.

static - Storage Class

static is the default storage class for global variables. The two variables below (count and road) both have a static storage class.

```
static int Count;
int Road;
{
printf("%d\n", Road);
}
```

static variables can be 'seen' within all functions in this source fila. At link time, the static variables defined bere will not be seen by the object modules that are brought in.

static can also be defined within a function. If this is done the variable is initalised at run time but is not reinitalized when the function is called. This inside a function static variable retains its value during vairous calls.

```
void func(void);
static count=10; /* Global variable - static is the default */
main()
{
while (count--)
{
func();
}
}
void func( void )
{
static i = 5;
i++;
printf("i is %d and count is %d\n", i, count);
}
```

**Q. 4. (b) what are the merits and demerits of static and dynamic memory aliocation techniquas ?**

**Ans. Static memory allocation :** In static memory allocation the compiler allocates the required memory space for a declared variable.By using the address of operator,the reserved address is obtained and this address may be assigned to a pointer variable.Since most of the

www.btechsamplepapers.in

declared variable have static memory,this way of assigning pointer value to a pointer variable is known as static memory allocation. memory is assigned during compilation time.

**Dynamic memory allocation :** Dynamic memory allocation uses functions such as malloc( ) or calloc( ) to get memory dynamically.If these functions are used to get memory dynamically and the values returned by these functions are assingned to pointer variables, such assignments are known as dynamic memory allocation.memory is assined during run time.

**Dynamic memory allocation :** The process of allocating memory at run time is known as dynamic memory allocation. Although c does not inherently have this facility there are four library routines which allow this function.

Many languages permit a programmer to specify an array size at run time. Such languages have the ability to calculate and assign during executions, the memory space required by the variables in the program. But c inherently does not have this facility but supports with memory management functions, which can be used to allocate and free memory during the program execution. The following functions are used in c for purpose of memory management.

| Function | Taak |
|---|---|
| malloc | Allocates memory requests size of bytes and returns a pointer to the Ist byte of allocated space |
| calloc | Allocates space for an array of elements initializes them to zero and returns a pointer to the memory |
| free | Frees previously allocated space |
| realloc | Modifies the size of previously allocated space. |

**Memory allocations proceas :** According to the concaptual view the program instructions and global and static variable in a permanent storage area and local area variables are stored in stacks. The memory space that is located between these two regions in available for dynamic allocation during the execution of the program. The free memory region is called the heap. The size of heap keeps changing when program is executed due to creation and death of variables that are local for functions and blocks. Therefore it is possible to encounter memory overflow during dynamic allocation process. In auch situationa, the memory allocation functions mentioned above will return a null pointer.

**Allocating a block of memory :** A block mf memory may be allocated using the function malloc. The malloc function reserves a block of memory of specified size and returns a pointer of type void. This means that we can assign it to any type of pointer. It takes the following form:

ptr=(cast-type*)malloc(byte-size);

ptr is a pointer of type cast-type the malloc returns a pointer (of cast type) to an area of memory with size byte-size.

**Example :**

x=(int*)malloc(100*sizeof(int));

On successful execution of this statement a memory equivalent to 100 times the area of int bytes is reserved and the address of the first byte of memory allocated is assigned to the pointer x of type int

**Allocating multiple blocks of memory :** Calloc is another memory allocation function that is normally used to request multiple blocks of storage each of the same size and then sets all bytes to zero. The general form of calloc is:

www.btechsamplepapers.in

ptr=(cast-type*) calloc(n,alam-siza);

The above statement allocates contiguous space for n blocks each size of elements size bytes. All bytes are initialized to zero and a pointer to the first byte of the allocated region is returned. If there is not enough space a null pointer is returned.

**Releasing the used space :** Compile time storage of a variable is allocated and released by the system in accordanca with its storage class. With the dynamic runtime allocation, it is our responsibility to release the space when it is not required. The release of storage space becomes important when the storage is limited. When we no longer need tha data we stored in a block of memory and we do not intend to use that block for storing any other information, we may ralease that block of memory for future use, using the free function.

free(ptr);

ptr is a pointer that has been created by using malloc or calloc.

**To alter the size of allocated memory :** The memory allocated by using calloc or malloc might be insufficient or excess sometimes in both the situationa we cen change the memory size already allocated with the help of the function realloc. This process is called reallocation of memory. The general statement of reallocation of memory is :

ptr=realloc(ptr,newsize);

This function allocates new memory space of size newsize to the pointer variable ptr ans returns a pointer to the first byte of the memory block. The allocated new block may be or may not be at the same region.

**Q. 4. (c) What are the different bit operators used in C ? Give an example of each .**

**Ans.** C provides six operators for bit manipulation; these may only be applied to integral operands, that is, char, short, int, and long, whether signed or unsigned.

| | |
|---|---|
| & | bitwise AND |
| \| | bitwise inclusive OR |
| ^ | bitwise exclusive OR |
| << | left shift |
| >> | right shift |
| ~ | one's complement (unary) |

The bitwise AND operator & is often used to mask off some set of bits, for example

```
n = n & 0177;
```

sets to zero all but the low-order 7 bits of n.

The bitwise OR operator | is used to turn bits on:

```
x = x | SET_ON;
```

sets to one in x the bits that are set to one in SET_ON.

The bitwise exclusive OR operator ^ sets a one in each bit position where its operands have different bits, and zero where they are the same.

One must distinguish the bitwise operators & and | from the logical operators && and ||, which imply left-to-right evaluation of a truth value. For example, if x is 1 and y ia 2, then x & y is zero while x && y is one.

The shift operatora << and >> perform left and right shifts of their left operand by the number of bit positions given by the right operand, which must be non-negative. Thus x << 2 shifts the value of x by two positions, filling vacated bits with zero; this is equivalent to

www.btechsamplepapers.in

multiplication by 4. Right shifting an unsigned quantity always fits the vacated bits with zero. Right shifting a signed quantity will fill with bit signs (``arithmetic shift'') on some machines and with 0-bits (``logical shift'') on others.

The unary operator ~ yields the one's complement of an integer; that is, it converts each 1-bit into a 0-bit and vice versa. For example

```
x = x & ~077
```

sets the last six bits of x to zero. Note that x & ~077 is independent of word length, and is thus preferable to, for example, x & 0177700, which assumes that x is a 16-bit quantity. The portable form involves no extra cost, since ~077 is a constsnt expression that can be evaluated at compile time.

As an illustration of some of the bit operators, consider the function getbits(x,p,n) that returns the (right adjusted) n-bit field of x that begins at position p. We assume that bit position 0 is at the right end and that n and p are sensible positive values. For example, getbits(x,4,3) returns the three bits in positions 4, 3 and 2, right-adjusted.

```
/* getbits: get n bits from position p */
unsigned getbits(unsigned x, int p, int n)
{
   return (x >> (p+1-n)) & ~(~0 << n);
}
```

The expression x >> (p+1-n) moves the desired field to the right end of the word. ~0 is all 1-bits; shifting it left n positions with ~0<<n places zeros in the rightmost n bits; complementing that with ~ makes a mask with ones in the rightmost n bits.

**Q. 5. (a) Simulate calculator using switch statement.**

**Ans.**

```
#include<stdio.h>
float add(float,float);
float sub(float,float);
float product(float,float);
float divide(float,float);
void main()
{
   float n1,n2;
   char sym,choice;
   printf("This Program is a program for calculator\n\n");
   scanf("%f%c%f",&n1,&sym,&n2);
   if(sym=='+')
      printf("\n%f",add(n1,n2));
   if(sym=='-')
      printf("\n%f",sub(n1,n2));
   if(sym=='*')
      printf("\n%f",product(n1,n2));
   if(sym=='/')
      printf("%f",divide(n1,n2));
   printf("\nDo you wish to continue[y/n]");
   scanf("%s",&choice);
   if(choice=='y'||choice=='Y')
      main();
}
float add(float m1,float m2)
```

www.btechsamplepapers.in

```
{
  return(m1+m2);
}
float sub(float m1,float m2)
{
  return(m1-m2);
}
float product(float m1,float m2)
{
  return(m1*m2);
}

float divide(float m1,float m2)
{
  return(m1/m2);
}
```

**Q. 5. (b) Implement Stack using Linked List.**

**Ans. Stack using linked list**

**PUSH(sptr H,INT x)**

in the operation push we have to insert the element at the next of the header in linked list.

to insert we have to create a node and insert element in the information field and we have to place that node in right position.take header in another pointer variable

sptr P=H;

create a new node

n=(sptr)malloc(sizeof(struct sptr));

if(n==NULL) "out of space"

else

n->info=x;

n->next=H->next;

H->next=n;

**POP**

int pop(sptr H)

in this pop(delete) operation we have to delete the element in the node next to header

check wether elements exists or not

if(H->next==NULL) "no elements in the stack"

else

store the element in a variable x

x=H->next->info

delete the node i.e.,change the links in the linked lists

P=H->next;

H->next=H->next->next;

we have to free the memory location by H->next;

delete(P);

Here is the code on how to implement stack using linked list using C programming. The Comments will help you get to know how the code works.

www.btechsamplepapers.in

**Code Starts here**

```
typedef struct node *nptr;
struct noda
{
int data;
nptr next;
};
nptr createhead(void);
void display(nptr s);
void push(nptr s,int x);
int pop(nptr s);
nptr createhead() /*Function to create head*/
{
nptr H;
H=(nptr)malloc(sizeof(struct node));
if(H!=NULL)
{
H->next=NULL;
return (H);
}
else
printf("\n\tOut of Space");
return;
}
void display(nptr s) /*Function to display the elements*/
{
nptr p;
p=s;
while(p->next!=NULL)
{
printf("%5d",p->next->data);
p=p->next;
}
}
void push(nptr s,int x) /*Function to push the elements*/
{
nptr temp;
temp=(nptr)malloc(sizeof(struct node));
if(temp==NULL)
{
printf("Out of Space");
return;
}
else
{
temp->data=x;
temp->next=s->next;
s->next=temp;
}
}
int pop(nptr s) /*Function to pop the elements*/
{
nptr temp;
int y;
if(s->next==NULL)
{
printf("Underflow on Pop");
return(-1);
}
else
{
y=s->next->data;
temp=s->next;
s->next=temp->next;
free(temp);
return(y);
}
}
```

**Q. 5. (c) Write a program C to multiply two matrices. Take the siza and alement of matric through keyboard.**

**Ans.**

```
void main()
{
  int m1[10][10],i,j,k,m2[10][10],add[10][10],mult[10][10],r1,c1,r2,c2;
  printf("Enter number of rows and columns of first matrix MAX 10\n");
```

www.btechsamplepapers.in

```
printf("Enter number of rows and columns of first matrix MAX 10\n");
scanf("%d%d",&r1,&c1);
printf("Enter number of rows and columns of second matrix MAX 10\n");
scanf("%d%d",&r2,&c2);
if(r2==c1)
{
   printf("Enter rows and columns of First matrix \n");
   printf("Row wise\n");
   for(i=0;i<r1;i++)
   {
      for(j=0;j<c1;j++)
         scanf("%d",&m1[i][j]);
   }
   printf("You have entered the first matrix as follows:\n");
   for(i=0;i<r1;i++)
   {
      for(j=0;j<c1;j++)
         printf("%d\t",m1[i][j]);
      printf("\n");
   }
   printf("Enter rows and columns of Second matrix \n");
   printf("Again row wise\n");
   for(i=0;i<r2;i++)
   {
      for(j=0;j<c2;j++)
         scanf("%d",&m2[i][j]);
   }
   printf("You have entered the second matrix as follows:\n");
   for(i=0;i<r2;i++)
   {
      for(j=0;j<c2;j++)
         printf("%d\t",m2[i][j]);
      printf("\n");
   }
   if(r1==r2&&c1==c2)
   {
      printf("Now we add both the above matrix \n");
      printf("The result of the addition is as follows:\n");
      for(i=0;i<r1;i++)
      {
         for(j=0;j<c1;j++)
         {
            add[i][j]=m1[i][j]+m2[i][j];
            printf("%d\t",add[i][j]);
         }
         printf("\n");
      }
   }
   else
   {
```

www.btechsamplepapers.in

```
    }
    printf("Now we multiply both the above matrix \n");
    printf("The result of the multiplication is as follows:\n");
    /*a11xA11+a12xA21+a13xA31a11xA12+a12xA22+a13xA32 a11xA13+a12xA23
                                        +a13xA33*/
    for(i=0;i<r1;i++)
    {
      for(j=0;j<c2;j++)
      {
        mult[i][j]=0;
        for(k=0;k<r1;k++)
        {
          mult[i][j]+=m1[i][k]*m2[k][j];
          /*mult[0][0]=m1[0][0]*m2[0][0]+m1[0][1]*m2[1][0]+m1[0][2]*m2[2][0];*/
        }
        printf("%d\t",mult[i][j]);
      }
      printf("\n");
    }
    getch();
  }
  else
  {
    printf("Matrix multiplication cannot be done");
  }
}
```

**Q. 6. (a) Show the usage of break and continue statement by taking an example.**

**Ans.** The break statement will immediately jump to the end of the current block of code.

The continue statement will skip the rest of the code in the current loop block and will return to the evaluation part of the loop. In a do or while loop, the condition will be tested and the loop will keep executing or exit as necessary. In a for loop, the counting expression (rightmost part of the for loop declaration) will be evaluated and then the condition will be tested.

**The break Statement :** We have already met break in the discussion of the switch statement. It is used to exit from a loop or a switch, control passing to the first statement beyond the loop or a switch.

With loops, break can be used to force an early exit from the loop, or to implement a loop with a test to exit in the middle of the loop body. A break within a loop should always be protected within an if statement which provides the test to control the exit condition.

**The continue Statement :** This is similar to break but is encountered less frequently. It only works within loops where its effect is to force an immediate jump to the loop control statement.

In a while loop, jump to the test stetement.

- In a do while loop, jump to the test statement.
- In a for loop, jump to the test, and perform the iteration.

**Like a break, continue should be protected by an if statement. You are unlikely to use it very often. To better understand the use of break and continue let us examine the following program :**

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

www.freesamplepapers.in

```
int main()
{
int value;
for(value = 5 ; value < 15 ; value = value + 1)
{
if (value == 8)
break;
printf("In the break loop, value is now %d\n", value);
}
for(value = 5 ; value < 15 ; value = value + 1)
{
if (value == 8)
continue;
printf("In the continue loop, value is now %d\n", value);
}
return 0;
}
```

**Q. 6. (b) Write an algorithm to sort the given list of integers.**

**Ans.**

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
 int item[100];
 int a, b, t;
 int count;
 /* read in numbers */
 printf("How many numbers? ");
 scanf("%d", &count);
 for(a = 0; a < count; a++)
   scanf("%d", &item[a]);
 /* now, sort them using a bubble sort */
 for(a = 1; a < count; ++a)
  for(b = count-1; b >= a; --b) {
   /* compare adjacent elements */
   if(item[ b - 1] > item[ b ]) {
    /* exchange elements */
    t = item[ b - 1];
    item[ b - 1] = item[ b ];
    item[ b ] = t;
   }
  }

 /* display sorted list */
 for(t=0; t<count; t++) printf("%d ", item[t]);
 return 0;
}
```

**Q. 6. (c) Discuss the usage of macro in C.**

**Ans. Macro Expansion**

Have a look at the following program.

```
#define UPPER 25
```

www.btechsamplepapers.in

```
{
int i ;
for ( i = 1 ; i <= UPPER ; i++ )
printf ( "\n%d", i ) ;
}
```

In this program instead of writing 25 in the **for** loop we are writing it in the form of UPPER, which has already been defined before **main( )** through the statement,

```
#define UPPER 25
```

This statement is called 'macro definition' or more commonly, just a 'macro'. What purpose does it serve? During preprocessing, the preprocessor replares every occurrenre of UPPER in the program with 25. Here is another example of macro definition.

```
#define PI 3.1415
main( )
{
float r = 6.25 ;
float area ;
area = PI * r * r ;
printf ( "\nArea of circle = %f", area ) ;
}
```

templates', whereas, 25 and 3.1415 are called their corresponding 'macro expansions'.

When we compile the program, before the source code passes to the compiler it is examined by the C preprocessor for any macro definitions. When it sees the **#define** directive, it goes through the entire program in search of the macro templates; wherever it finds one, it replaces the macro template with the appropriate macro expansion. Only after this procedure has been rompleted is the program handed over to the compiler.

In C programming it is customary to use capital letters for macro template. This makes it easy for programmers to pick out all.the macro templates when reading through the program.

**Q. 7. (a) What is the use of header file ? Discuss.**

**Ans.** The preprocessor rommand for file inclusion looks like this:

```
#include "filename"
```

and it simply causes the entire contants of **filename** to be inserted into the source code at that point in the program. Of course this presumes that the file being included is existing. When and why this feature is used ? It can be used in two cases:

(a) If we have a very large program, the code is best divided into several different files, each containing a set of related functions. It is a good programming practice to keep different sections of a large program separate. These files are **#included** at the beginning of main program file.

(b) There are some functions and some macro definitions that we need almost in all programs that we write. These commonly

needed functions and macro definitions can be stored in a file, and that file can be included in every program we write, which would add all the statements in this file to our program as if we have typed them in.

It is common for the files that are to be included to have a .h extension. This extension stands for 'header file', possibly because it contains statements which when included go to the head of your program. The prototypes of all the library functions are grouped into different categories and then stored in different header files. For example prototypes of all mathematics related functions are stored in the header file 'math.h', prototypes of console input/output functions are stored in the header file 'conio.h', and so on.

Actually there exist two ways to write **#include** statement. These are:

```
#include "filename"
#include <filename>
```

www.techsamplepapers.in

#include <filename>

**Q. 7. (b) Write a program to show the usage of atructure in C.**

**Ans.** A structure in C is a collection of items of different types. You can think of a structure as a "record" is in Pascal or a class in Java without methods.

These data types may or may not be of the same type. The following example illustrates the use of this data type.

```
main( )
{
struct book
{
char name ;
float price ;
int pages ;
} ;
struct book b1, b2, b3 ;
printf ( "\nEnter names, prices & no. of pages of 3 books\n" ) ;
scanf ( "%c %f %d", &b1.name, &b1.price, &b1.pages ) ;
scanf ( "%c %f %d", &b2.name, &b2.price, &b2.pages ) ;
scanf ( "%c %f %d", &b3.name, &b3.price, &b3.pages ) ;
printf ( "\nAnd this is what you entered" ) ;
printf ( "\n%c %f %d", b1.name, b1.price, b1.pages ) ;
printf ( "\n%c %f %d", b2.name, b2.price, b2.pages ) ;
printf ( "\n%c %f %d", b3.name, b3.price, b3.pages ) ;
}
```

And here is the output...

```
Enter names, prices and no. of pages of 3 books
A 100.00 354
C 256.50 682
F 233.70 512
```

**Q. 7. (c) Write a program in C to find the factorial of a given number by recursion.**

**Ans.**

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
printf("hello world  this is my program to find factorial ");
int a,b,c,n,i;
b=1;
printf("\n enter the no to find factorial ");
scanf("%d",&n);
printf("\n factorial as follows ");
for(i=n;i>1;i--)
{ c=b*i;
  b=c;
}
printf("%d",c);
getch();
}
```

www.btechsamplepapers.in