C Programming and Data Structures
(May/June 2006)

SET 4

1. (a) **What is meant by operator precedence? What are the relative Precedence of the arithmetic operators ?**

Each operator in 'C' has a precedence associated with it. This Precedence is used to determine how an expression involving more than one operator is evaluated. There are distinct levels of precedence and an operator may belong to one of these levels. The relative precedence of arithmetic operators can be given from two distinct levels, they are

HIGH PRIORITY: * , / , %
LOW PRIORITY: +, –

The basic evaluation procedure includes two left to right passes during the first pass , the high priority operators are applied as they are Encountered during the second pass, the low priority operators are applied as they are encountered.

(b) **What is the associativity of the arithmetic operators?**

Arithmetic operators are *,/, +, –, %.

Operator	Description	Associativity
+	Addition	Left to right
–	Subtraction	Left to right
*	Multiplication	Left to right
/	Division	Left to right
%	Modulus	Left to right

(c) **How can the value of an expression be converted to a different data Types?**

The conversion of data from one form to another form is called type conversion.
Automatic type conversion:

If the operands are different types, the ‘lower’ type is automatically converted to the ‘higher’ type . The result is of the higher type.

Ex:

```
int female;
float male, ratio;
ratio= female/male;
```

Casting of values:

The values of an expression can be converted into different data types by casting the value. Consider the following example:

Since female and male are declared as integers in program the decimal part of the result of the division would be lost and the ratio would represent a wrong figure. This problem can solved by converting one of the variables locally to floating point.

```
int female, male;
float ratio;
ratio=(float)female/male;
```

The result will be converted into **float**. The process of such a local conversion is known as casting of value. The general form is:

(*data type name*) expression.

(d) What are unary operator? Explain example for each.

The unary operators are the operators which require only one operand to perform the operation.

The unary operators are

(1) ! –logical not

Eg:

a	!a
T	F
F	T

(2) Short hand assignment operators:

Eg:

a+ =	works as a = a+1
a– =1	works as a = a-1
a*=1	works as a = a*1
a/=1	works as a = a/1

(3) Increment and decreament operators:

Eg: let m = 5 and result be stored in y.

```
m = 5
m++; //m = m+1//
```

resultant m = 6

```
m = 5;
–m;  //m=m–1//
```

Resultant m = 4.

(4) Bitwise left shift operator, bitwise right shift operator

(a) suppose a =4

A<<2 . if left shifts the binary bits twice.

So a = 00010000

(b) suppose a>>2 . if right shifts the binary bits twice.
so a = 00000001

(5) size of operator.
Eg: suppose a is declared as integer. The size of a is

```
Int a;
x = size of (a);
x = 2.
```

2. (a) In what way array is different from an ordinary variable?

An array is a collective name given to a group of similar elements whereas a variable is any entity that may change during program execution.

An array is a variable that is capable of holding many values where as an ordinary variable can hold a single value at a time. For example, an array initialization would be

Int number [8];

for which there are 8 storage locations reserved where 8 different values can be stored belonging to the same type.

An ordinary variable initialization would be

Int number; & only one storage location is reserved and can hold only one value at a time.

(b) What conditions must be satisfied by the entire elements of any given Array?

Elements of an array can be initialized at the time & place of their declaration. Consider an array with it's elements.

Int a [4] = {1,2,3,4};

In the above example, 4 elements are stored in array 'a' The array elements are stored in continuous memory locations.

The array elements are read from '0' i.e a [0], is assigned the value '1', similarily a [1] is arranged the value '2',a [2] is assigned '3'.

The condition that needs to be satisfied is that, all the elements of the array must be of the same data type under which the array is declared

(c) What are subscripts? How are they written? What restrictions apply to the Values that can be assigned to subscripts?

Subscript of an array is an integer expression, integer constant or integer variable like 'i' ,'n' etc that refers to different elements in the array. Consider a statement

int a[5];

here, array subscripts start at 0 in c-language. Therefore the elements are a [0], a [1], ………a [4].

The values that can be assigned to the subscripts always must start from '0' and it ends at "size of array-1" for example consider the array

int a[5]

here the range of subscripts starts from '0' and it ends at '4'.

(d) What advantages is there in defining an array size in terms of a symbolic constant rather than a fixed integer quantity?

The advantage of defining the size of an array using symbolic constant instead of fixed numbers is that, by defining the symbolic constant at the beginning of the program, it allows us to know the maximum size of the array and in future if we want to change the size of the array, it can be easily done by modifying the definition of the symbolic constant which gets reflected throughout the program.

(e) Write a program to find the largest element in an array.

/*program to find largest element in an array */

```
#include<stdio.h>
#include<conio.h>
main()
{
int I,a[10],l,n;
printf(“enter number of elements in the array”);
scanf(“%d”,&n);
printf(“enter the array elements”);
for(i=0;i<n;i++)
scanf (“%d”,&a[i]);
l=a[0];
for (i=o;i<n;i++)
{
if (l<a[i])
l=a[i];
}
printf(“the largest element is %d”,l);
getch();
}
```

3. (a) Write a 'C' program to compute the sum of all element stored in an array Using pointers.

```
/*program to compute sum of all elements stored in an
array */
#include<stdio.h>
#include<conio.h>
main()
{
int a[10],I,sum=0,*p;
printf("enter 10 elements \n");
for(i=0; i<10;i++)
scanf ("%d", & a[i]);
p = a;
for(i = 0; i<10; i++)
{
sum = sum*p;
p++;
}
printf("the sum is % d",sum);
getch();
}
```

(b) Write a 'C' program using pointers to determine the length of a character String.

```
/*program to find the length of a char string */
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
Char str [20].*p;
Int l=0;
printf("enter a string \n");
scanf (" % s",str);
p=str;
while(*p!='\0')
{
```

```
l++;
p++;
}
printf(“the length of the given string is %d”,l);
getch();
}
```

4. (a) Explain the different ways of passing structure as arguments in functions .

There are different ways of passing structure as an argument to functions.
They are

1. Passing structure members individually: each member of the structure can be passed individually as arguments in the function call and retained through return statement.

Ex:
```
#include<stdio.h>
 struct x
 {
 int a;
 char b;
 } P;
 main()
 {
 struct x,m;
 printf(“enter a,b values”);
 scanf(“%d%c”,&n.a,&m.b) ;
 fun(i,c)
 int i ;

 char c ;

 {
 printf(“a = % d\t b=% d ‘’, I,c);

 }
```

2. A copy of complete structure can be passed as argument using this method, a copy of entire structure is passed to function but any modifications made to it will not reflected in the called function.
eg: consider a structure x,which has two members, one integer type and the other character type. We can pass the entire structure to a function as follows.

```
#include<stdio.h>
struct x
{
int a;
char b;
}
main()
{
struct x,m;
printf(“enter values of a & b”);
scanf(“%d%c”,&m.a,&m.b);
fun(m);
}
fun(y)
struct x y ;
{
printf(“a=%d b=%c”,y.a,y.b) ;
}
```

3. By passing the address of the structure to the called function. This method is more efficient than the above method because the structure need not be returned to called function. In this approach, the address location of the structure is passed to called function.

ex:

```
#include<stdio.h>
struct marks
{
float avg;
int m[3],tot;
}*p
main()
{
struct marks student;
printf(“enter 3 subject marks”);
for(“i=0;i<2;i++)
scanf(“%d”,&student.m[i]);
total(&student,3);
printf(“%d is total and %f is avg”,student.total ,student.avg);
}
total(student ,n)
struct marks *student;
int n;
{
int I,t=0;
for(i=0;i<n;i++)
t=t+m[i];
student—tot=t;
student—avg=t/n;
}
```

(b) Write a ‘C’ program using pointers to determine the length of a character String.

/*program to illustrate the method of sending the entire structure as a parameter to function */

```
#include<stdio.h>
struct marks
{
int m[3],tot;
float avg;
}
main()
{
struct marks mk1;
void total(struct marks,int);
mk1.m[0]=50;
mk1.m[1]=70;
mk1.m[2]=80;
mk1.tot=0;
mk1.avg=0;
mk1=total(mk1,3);
printf(“total=%d,average=%f”,mk1.tot,mk1.avg);
}
struct marks total(struct marks mk2,intn)
{
int t=0;
for (i=0;i<n;i++)
t = t+m[i];
mk2.to t= t;
mk2.avg=t/n;
return(mk2);
}
```

5. (a) Distinguish text mode and binary mode operation of file.

Text Mode	Binary Mode
1. In this mode file is opened For one purpose.	1. We can perform binary Operations by opening a file in this mode.
2. "r" opens the file for reading Only.	2. "r+" opens the existing File for both reading and writing.
3. "w" opens the file for writing Only.	3. "w+" is same as write mode but for both read and writing .
4. "a" opens the file for appending data to it.	4. "a+" same as 'a' except for both reading and writing.

(b) Write a program to open a pre-existing file and add information at the End of a file. Display the contents of the file before and after appending.

```
#include<stdio.h>
#include<conio.h>
main()
{
char x;
fILE *f;
f=fopen("data",''r");
while(x=getc(f)!=EOF)
printf("%c",x);
fclose(f);
```

```
 f = fopen(“data”,‘‘a”);
 while((x=getchar())!=EOF)
 putc(x,f);
 fclose(f);
 f = fopen(“data”,‘‘r”);
 while((x=getc(f))!=EOF)
 printf(“%c”,x);
 getch();
}
```

6. Write a ‘C’ program using pointers to implement a stack with all the operations.

```
 #include<stdio.h>
 #include<conio.h>
Int size;
struct stack
{
 Int a[max];
Int top;
};
void stk(struct stack *st)
{
 st->top = –1;
}
void push(struct stack *st,int num)
{
 If(st->top == size–1)
 {
 printf(“\n over flow\n”);
 return;
 }
 st->top++;
 st->a[st->top]=num;
 }
Int pop(stuct stack *st)
 {
 Int num;
 If(st->top==-1)
 { printf(“stack is under folw\n”);
 return NULL;
 }
 num=st->a[st->top];
 st->top—;
 return num;
}
void display(struct stack *st)
 {
Int i;
for(i=st->top;i>=0;i—)
 printf(“\n %d\t”,st->a[i]);
 }
void main()
{
 Int d,i,n;
struct stack *ptr;
 do
{
 printf(“\n menu items\n1.push \t 2.pop\t3.display\t
4.exit\n”);
 printf(“enter your choice:”);
 scanf(“%d”,&i);
 switch(i)
 {
 case 1: printf(“enter an element:”);
 scanf(“%d”,&n);
 push(&ptr,n);
 break;
 case 2: d=pop(&ptr);
 printf(“\n deleted item %d”,d);
 break;
     case 3: printf(“elements of the stack are \n”);
             display(&ptr);
```

```
                break;
        case 4: exit (0);
        default: printf("invalid choice");
 }
 }
getch();
}
```

7. Write a routine SPLIT() to split a singly linked list into two lists so that all elements in odd position are in one list and those in even position are in another list.

```
/* Implementation of linked list including split operation*/
#include<stdio.h>
#include<alloc.h>
#define null 0
struct linked_list
{
int data;
struct linked_list *next;
}*first,*fresh,*ptr,start1,ptr1,start2,ptr2;
typedef struct linked_list node;
main()
{
int ch,a;
clrscr();
while(1)
{
printf("\n MAIN MENU \n");
printf("1.insert element\n");
printf("2.delete element\n");
printf("3.view contents\n");
printf("4.split\n");
printf("5.exit from program\n");
printf("enter your choice\n");
scanf("%d",&ch);
switch(ch)
{
```

```
case 1:

    fresh=malloc(sizeof(node));

    printf(“enter the element to be inserted\n”);

    scanf(“%d”,&fresh->data);

    printf(“where do you want to insert\n”);

    printf(“1.begining\n2.end\n3.middle\n”);

    scanf(“%d”,&ch);

    switch(ch)

    {

    case 1:

          lbegin();
          break;

    case 2:

          lend();
          break;

    case 3:

          printf(“enter position\n”);
          scanf(“%d”,&a);
          lmiddle(a);
          break;

    }

          break;

case 2:

    printf(“where do you want to delete\n”);
    printf(“1.begining\n2.end\n3.required position\n”);
    scanf(“%d”,&ch);
    switch(ch)
    {
      case 1:
      Dbegin();
      break;
      case 2:
      Dend();
      break;
case 3:
      printf("enter position\n");
      scanf("%d",&a);
      Dmiddle(a);
      break;
      }
      break;
case 3:
      clrscr();
      break;
case 4:
      split();
      break;
case 5:
      exit(0);
default:
      printf("wrong choice\n");
      break;
}
}
getch();
}
/* Insertion function */
Ibegin()
{
if(first==null)
```

```
{
first=fresh;
first->next=null;
}
else
{
fresh->next=first;
first=fresh;
}
}
lend()
{
if(first==null)
{
printf("list is empty, inserted element is the last/first\n");
first=fresh;
first->next=null;
}
else
{
ptr=first;
while(ptr->next!=null)
ptr=ptr->next;
ptr->next=fresh;
fresh->next=null;
}
}
lmiddle(int n)
{
int i;
if(first==null)
{
printf("list is empty, inserted element is the last/first\n");
first=fresh;
first->next=null;
}
else
{
ptr=first;
for(i=1;i<n;i++)
ptr=ptr->next;
fresh->next=ptr->next;
ptr->next=fresh;
}
}
/* Deletion function */
Dbegin()
{
ptr=first;
if(ptr->next==null)
{
if(first==null)
{
puts("list is empty,deletion not possible");
return;
}
else
{
printf("list contains only one element and now it is empty
due to deletion\n");
first=null;
free(first);
}
}
else
first=ptr->next;
free(ptr);
}
Dend()
{
ptr=first;
```

```
if(ptr->next==null)
{
if(first==null)
{
puts("list is empty,deletion not possible");
return;
}
else
{
printf("list contains only one element and now it is empty
due to deletion\n");
first=null;
free(first);
}
}
else
{
while(ptr->next->next!=null)
ptr=ptr->next;
free(ptr->next->next);
ptr->next=null;
}
}
Dmiddle(int n)
{
int i;
ptr=first;
if(ptr->next==null)
{
if(first==null)
{
puts("list is empty,deletion not possible");
return;
}
else
```

```
{
printf("list contains only one element and now it is empty
due to deletion\n");
first=null;
free(first);
}
}
else
{
for(i=1;i<n-1;i++)
ptr=ptr->next;
fresh=ptr->next;
ptr->next=ptr->next->next;
fresh->next=null;
free(fresh);
}
}
view()
{
ptr=first;
if(ptr->next==null)
{
if(first==null)
{
puts("list is empty");
return;
}
else
{
printf("%d",first->data);
}
}
else
{
for(ptr=first;ptr->next!=null;ptr=ptr->next)
```

```
printf("%d->",ptr->data);
printf("%d",ptr->data);
}
}
/* split function */
split()
{
ptr=first;
ptr1=start1;
ptr2=start2;
while(ptr->next!=null)
{
    if(start1==null)
    {
    start1=ptr;
    ptr=ptr->next;
    start1->next=null;
    }
    else
    {
    ptr1->next=ptr;
    ptr1=ptr;
    ptr=ptr->next;
    ptr1->next=null;
    }
    if(start2==null)
    {
    start2=ptr;
    ptr=ptr->next;
    start2->next=null;
    }
    else
    {
    ptr2->next=ptr;
    ptr2=ptr;
```

```
        ptr=ptr->next;
        ptr2->next=null;
        }
}
/* printing the two lists */
        ptr1=start1;
        ptr2=start2;
        printf("EVEN LIST IS\n");
        while(ptr1->next!=null)
        {
        printf("%d\t",ptr1->data);
        ptr1=ptr1->next;
        }
        printf("ODD LIST IS\n");
        while(ptr2->next!=null)
        {
        printf("%d\t",ptr2->data);
        ptr2=ptr2->next;
        }
}
```

8. **Write a 'C' program to sort a given list of elements using tree sort and discuss its time complexity.**

```
 #include<stdio.h>
 #include define MAXSIZE 50
 Void heapsort(int elements[ ],int maxsize);
 Void heap(int elements[ ], int root, int leaf);
 Int elements[MAXSIZE],maxsize;
 Main();
{
 int i;
 printf("\n enter no of elements:");
 scanf("%d",&maxsize);
printf("enter the elements to be sorted\n");
for(i=0;i<maxsize;i++)
{
```

```
Scanf("%d",&elements[i]);
}
Printf("array before sorting is \n");
For(i=0;i<maxsize;i++)
Printf("%d",elements[i]);
Heapsort(elements,maxsize);
Printf("\narray after sorting is \n");
For(i=0;i<maxsize;i++)
Printf("%d",elements[i]);
}
Void heapsort(int elements[],int maxsize)
{
Int i,temp;
For(i=(maxsize%2)-1;i>0;i—)
{
Temp=elements[0];
Elements[0]=elements[i];
Elements[i]=temp;
heap(elements,0,i-1);
}
}
void heap(int elements[],int root, int leaf)
{
Int flag, max, temp;
Flag=0;
While((root*2<=leaf)&&(!flag))
{
If(root*2==leaf)
Max=root*2;
Else if(elements[root*2]>elements[(root*2)+1])
Max=root*2+1;
If(element[root]<element[max])
{
Temp=elements[root];
Elements[root]=elements[max];
Elements[max]=temp;
Root=max;
}
Else
Flag=1;
}
}
```

TIME COMPLEXITY:-

A complete binary tree which is a heap with ‘m’ nodes will have ‘$\log(m+1)$’ levels. Thus we may have an efficiency of $O(m\log m)$. So, for heapsort both the average case and worst case time complexity are of the order of ‘$m\log m$’.