C Programming and Data Structures
(May/June 2006)

SET 3

1. (a) **What is the purpose of break statement?**

Some times it is convenient to be able to exit from a loop other than by testing at the top or bottom. The 'break' statement provides an early exit from 'for', 'while' and 'do-while' loops. This can be explained as follows.

Generally, looping job is executed until the given condition of loop statement is false. When the condition is false the loop is terminated. If we give ''break'' statement, the job will be terminated in the middle, irrespective to the given condition in looping.

For Example,

```
Void main ( )
   {
Int i;
For (i=1; i<=20; i++)
   {
       Printf ("%d", i);
       If (i==5)
       Break;
   }
}
```

This will print the numbers from 1 to 5 but not 1 to 20.

(b) **Suppose a break statement is included within the innermost of several nested control statements. What happens when break statement is executed?**

If a ''break'' statement included within the innermost of several nested control statements, are executed, the ''break'' would only exit from the loop counting it.

That is ''break'' will exit only a single loop and the statements which are in the next outer loops of the program are executed. But it does not come to the end of program.

This can be seen with following example.

```
For (------------)
{
   Statement(s);
```

```
          For (------------)
          {
              Statement (s);
              If (condition)
              Break;
          }
              Statement(s);
      }
```

Here, the break is in the inner ‘‘for’’ loop. When it is executed, it comes to outer ‘‘for’’ loop.

(c) Write a program to print the multiplication table up to with proper format.

```
          /* program to print multiplication table */
          #include<stdio.h>
          #include<conio. h>
          Void main()
              {
                  Int I, m, n;
printf(“enter no for which you want to print
Multiplication table and also enter upto how many
Values do you want to print it”);
scanf(“%d%d”,&n,&m);
for(i=1; i<=m;; i++)
 {
 printf(“%d*%d=%d”,n,i,n*i);
 printf(“\n”);
 }
}
```

2. (a) Write a program to demonstrate passing an array argument to a function. Consider the problem of finding the largest of *N* numbers defined in an array.

```
#include<stdio.h>

#include<conio.h>
int largest(int x[], int m)
```

```
{
 int i, max = x[0];
for(i=0; i<m; i++)
 if(x[i]>max)
 max = x[i];
 return(max);
 }
void main()
{
      int A[20], n, 1;
      printf(“Enter no. Of values”);
      scanf(“%d”, & n);
      printf(“Enter values”);
      for(i = 0;i<n; i++)
      scanf(“%d”, &A[i]);
      1 = largest(A, n);
      printf(“largest value in array is %d”, 1);
}
```

(b) Write a recursive function power (base, exponents) that when invoked returns base exponent.

```
#include<stdio.h>

int power(int base, int exp)
{
      int i = 0, p = 1;
      while(exp>0)
      {
      p = p*base;
      }
      return(p);
}
void main()
```

```
{
    int b, e, 1;
    printf(“Enter base & exponent”);
    scanf(“%d%d”, &b, &e);
    1=power(b, e);
    printf(“%d^%d = %d”, b, e, 1);
}
```

3. The roots of a quadratic equation of the form $ax^2+bx+c = 0$ are given by the following equations:

$$X_1 = -b + \sqrt{\mathrm{b}^2} -4ac$$

$$X_2 = -b - \sqrt{\mathrm{b}^2} -4ac$$

The roots of a quadratic equation of the form $ax^2+bx+c = 0$ are given by the following equations:

$$X_1 = -b+\sqrt{b^2} -4ac/2a$$

$$X_2 = -b-\sqrt{b^2} -4ac/2a$$

Write a function to calculate the roots. The function must use two pointer parameters, one to receive the coefficients *a*, *b* and *c* and the other to send roots to calling function.

```
#include<stdio.h>
#include<math.h>
Roots(p,q)
float *p,*q;
{
*q=(–(*(p+1)+sqrt((*(p+1))*(*(p+1))–4*(*p)*(*(p+2))))/
(2*(*p));
*(q+1)=(–(*(p+1)-sqrt((*(p+1))*(*(p+1))–4*(*p)*(*(p+2))))/
(2*(*p));
}
void main()
{
    float A[3], R[2];
    int i;
    printf(“Enter values for a, b, c”);
```

```
        for(i=0; i< = 2; i++)
        scanf("%f",A+i);
        Roots(A, R);
        printf("root1 = %f", *(R+0));
        printf("root2=%f", *(R+1));
}
```

4. (a) **Write a program to create an array of student structure objects and to find the highest marks scorer. The fields in the student structure are: name, age and marks.**

```
#include<stdio.h>
struct student
{
        char name[15];
        int age;
        float marks;
}X[15];
void main()
{
        int m, n;
        float max;
        printf("Enter no. of students");
        scanf("%d",&n);
        printf("Enter student name, age,marks");
        for(i = 0;i <m; i++)
        scanf("%s%d%f",X[i].name, &X[i]. age, &X[i]. marks);
        max=X[0].marks; n = 0;
        for(i = 0; i<m; i++)
{
        if(max<X[i].marks)
        n = i;
}
printf("highest marks scorer is %s", X[n].name);
}
```

(b) How are structure elements stored in memory?

1. Members of structure themselves are not variables. So, they do not occupy any memory until they are associated with structure objects.
2. When an object is created, a memory is created equal to the total memory occupied by all its members individually.
3. The elements are stored in consecutive memory blocks in main memory.

5. Write a program to read a 'C' program file and count the following in the complete 'C' program.

(a) Total number of statements

(b) Total number of opening brackets.

```
#include<stdio.h>
void main()
{
    FILE *f1;
    int n = 0;
    char c;
    /*To count total number of statements*/
    f1=fopen(“file1”,“r”);
    while((c = getc(f1))! = EOF)
    {
        if(c ==‘;’)
        n++;
    }
    printf("No. of statements = %d",n);
    fclose(f1);
    /*To count total number of opening brackets"*/
    n = 0;
    f1 = fopen(“file1”,“r”);
    while((c = getc(f1))! = EOF)
    {
    if(c = =’{’)
    n++;
}
printf(“No. of opening brackets = %d”, n);
```

```
    fclose(f1);
}
```

6. Write a ‘C’ program for implementation of various operations on circular queue.

```
#include<stdio.h>
#define size 4
int A[10], n=0, front = 0, rear = -1;
Enque(e)
int e;
{
       if((rear+1)%size==front&&n==size)
       printf(“Overflow......”);
       else
       {
       rear = (rear+1)%size;
       A[rear] = e;
       n++;
       }
}
int dequeue()
{
 int a;
       if(front==(rear+1)%szie&&n==0)
{
printf("Underflow");
return(0);
}
else
{
a=A[front];
printf(“Deleted item is %d”,a);
n--;
```

```
          front = (front+1)%size;
          return(a);
          }
          }
Traversal()
{
 int i;
if((rear+1)%size==front&&n==0)
 printf("Circular queue is empty");
else
{
          printf(“elements in circular queue are..”);
          for(i = front;i! = rear;i = (i+1)%size)
          printf(“%d”,A[i]);
          printf(“%d”,A[i]);
}
}
main()
{
          int i, n, ch;
          do
          {
          printf(“\t1.Enqueue\t2.Dequeue\t3.Traversal\t4.Exit”);
          printf(“Enter ur choice”);
          scanf(“%d”,&ch);
          switch(ch)
          {
          case 1:
                printf(“Enter element to enqueue”);
                scanf(“%d”,&n);
                Enqueue(n);
```

```
        break;
case 2:
        dequeue();
        break;
case 3:
        Traversal();
        break;
case 4:
        exit(0);
default:
        puts(“Invalid choice”);
}
}while(1);
}
```

7. **Represent a doubly linked list using an array. Write routines to insert and delete elements for this representation.**

```
 /* Double linked list using array*/
#include<stdio.h>
#include<alloc.h>
     struct node
     {
     int data;
     struct node *next, *prev;
 }*1, *new, *start;
 Insert(x)
 int x;
 { int loc, ch;
     if(start==NULL)
        {
        start = malloc(sizeof(struct node));
        start -> data = x;
        start -> next = NULL;
        start -> prev = NULL;
        }
else
 {
        new = malloc(sizeof(struct node));
        new -> data = x;
printf("enter your choice 1.insertion at starting 2.ending
3.middle");
        scanf("%d",&ch);
        if(ch==1)
        {
        new - >prev = NULL;
        new -> next = NULL;
        start -> prev = new;
        start  = new;
 }
 else if(ch==2)
        {
        for(l = start; l-> next! = NULL;l = l->next;);
        l -> next = new;
         new -> prev = NULL;
         new -> next = NULL;
 }
 else
        {
        printf("enter location to insert:");
        scanf("%d",&loc);
        for(1 = start,i = 1;i < loc–1; i++)
        l = l - >next;
if(l==NULL)
 {
puts("\a invalid location");
break;
 }
else
{
new -> next = 1 - > next;
new -> prev = 1;
1 -> next = new;
}
}
}
}
Delete()
{
    int ch, i = 1, loc;
    if(start == NULL)
    {
    printf(“empty”);
    return;
    }
    if(start -> next = NULL&&start -> prev == NULL)
     {
    printf(“deleted item %d”, start -> data);
    start = NULL;
    return;
 }
printf(“1.deletion at starting 2.ending 3.req postion”);
printf(“enter your choice”);
scanf("%d", &ch);
if(ch == 1)
{
p = start;
```

```
start = start -> next;
start -> prev = NULL;
printf(“deleted item id %d”, p -> data);

free(p);
}
else if (ch == 2)
{
for(1 = start; 1 -> next -> next! = NULL; 1=1 -> next)
{
    p = 1- > next;
    1 -> next = NULL;
printf(“deleted item is %d”, p -> data);
free(p);
}
else
    {
printf(“enter location to delete:”);
scanf(“%d”, & 1oc);
for(1 = start; i< 1oc –1; 1 = 1 -> next)
    i++;
if(1 == NULL)
{
    puts (“invalid location”):
return;
}
    p = 1 -> next;
    1 -> next = 1 -> next -> next;
    1 -> next -> prev = 1;
    printf(“deleted item is%d”,p->data);
    free(p);
    }
```

```
        }
main()
{
        int ch, n;
        do

        {

        printf("\n1.Insert 2.Delete 3.Exit");
        printf("Enter ur choice");
        scanf("%d",&ch);
        switch(ch)
        {
case 1:
        printf("Enter element to insert");
        scanf("%d",&n);
        Insert(n);
        break;
case 2:
        Delete();
        break;
case 3:
        exit(0);}
}while(1);
}
```

8. Write an algorithm for two-way merge sort. Analyze its time complexity.

```
#include<stdio.h>
        #define max 20
int a[max], b[max];
main( )
{
int i, n;
printf("enter the value of n");
```

```
scanf(“%d”, &n);
for(i = 0;i < n; i++)
scanf(“%d”, &a[i]);
mergesort(0, n);
printf(“sorted list is”);
for(i = 0;i < n; i++)
printf(“%d”, a[i]);
}
/* merge sort function */
mergesort(low,high)
int low, high;
{
int mid;
if(low < high)
{
mid = (low+high)/2
merge sort(low, mid);
mergesort(mid+1, high);
merge(low, mid, high);
}
}
merge(low, mid, high)
int low, mid, high;
{
int i, h, j;
h = low, i = low, j = mid+1;
while((h < = mid)&&( j < = high))
{
if(a[h]<=a[j])
{
b [i] = a[h];
h = h+1;
}
else
{
b[i] = a[j];
j = j+1;
}
i = i+1;
}
if(h > mid)
for(k = j;k < high; k++)
{
b[i] = a[k];
i++;
}
else
for(k = h; h < mid; h++)
{
b[i] = a[k];
i++;
}
for(k = low; k < high; k++)
a[k] = b[k];
}
```

TIME COMPLEXITY: If the time for the merging operation is proportional to n, then the computing time for merge sort is described by the recurrence relation.

$$T(n) = \{ \; a \quad n = 1, a \text{ a constant}$$

$$aT(n/2)+cn \quad n > 1 \text{ , } c \text{ a constant}$$

When n is a power of 2 , $n = 2\hat{}k$, we can solve this equation by successive substitutions:

$$
\begin{aligned}
T(n) &= 2(2T(n/4)+cn/2)+cn \\
&= 4T(n/4)+2cn \\
&= 4(2T(n/8)+cn/4)+2cn \\
&: \\
&: \\
&= 2\hat{}kT(1)+kcn \\
&= an + cn\log n
\end{aligned}
$$

It is easy to see that if $2\hat{}k < n <= 2\hat{}(k+1)$, then $T(n) <= T(2\hat{}k+1)$.Therefore,

$$T(n) = O(n\log n).$$