C Programming and Data Structures
(May/June 2006)

SET 2

1. Write about space requirements for variables of different data types.

1. C language is rich in its data types. The variety of data types available allows the programmer to select the type appropriate to the needs to the application as well as the machine.

   ANSI C supports four classes of data types:

1. Primary data type. Example, Integral type, Floating point type.
2. User-defined data type. Example, Main.
3. Derived data type. Example, arrays, structures, functions.
4. Empty data set.

All C compilers support four primary data types, namely (int), character (char), floating point (float) and double-precision floating point double.

The primary data types in C are tabulated as follows:

PRIMARY DATA TYPES		
INTEGRAL TYPE		
INTEGER		CHARACTER
SIGNED TYPE	UNSIGNED TYPE	Signed char
int	Unsigned int	Unsigned char
Short int	Unsigned short int	
Long int	Unsigned long int	

FLOATIING DATA TYPE		
Float	Double	Long Double

Size and Range of Basic Data Types	
DATA TYPE	RANGE OF VALUES
char	–128 to 127
int	–32,768 to 32,767
float	3.4 e –38 to 3.4 e +38
double	1.7 e –308 to 1.7 e +308

INTEGER TYPES: Integers are whole numbers with a range of values supported by a particular machine. Generally, integers occupy one word of storage, and word size of machines vary (typically, 16 or 32 bits) the size of an integer that can be stored depends on the computer. A signed integer uses one bit for sign (usually MSB) and 15 bits for magnitude of the number.

C has three classes of integer storage, namely short, int, long int, in both signed and unsigned forms. Usually the range of unsigned integer numbers will be from 0 to 65,535.

We declare long and unsigned integers to increase the range of values.

Size and Range of Data Types on a 16-Bit Machine.

Type	Size(bits)	Range
Char or signed char	8	–128 to 127
Unsigned char	8	0 to 255
Int or signed int	16	–32,768 to 32,767
Unsigned int	16	0 to 65,535
Short int or	8	–128 to 127
Signed short int		
Signed short int		
Unsigned short int	8	0 to 255
Long int or signed Short int	32	–2,147,483,648 to 2,147,483,647
Unsigned long int	32	0 to 4,294,967,295
Float	32	3.4E–38 to 3.4E+38
Double 1.7E+308 to	64	1.7E–308 to
Long double 1.1E+4932	80	3.4E–4932 to

FLOATING POINT TYPE: Floating point type numbers are defined in C by the keyword 'float'. When the accuracy provided by a float number is not sufficient the type double can be used to define the number.

CHARACTER TYPES: A single character can be defined as a character type data. Characters are usually stored in 8 bits. The qualifier signed or unsigned may be explicitly applied to character data type.

2. The annual examination is conducted for 50 students for three subjects. Write a program to read the data and determine the following:

(a) Total marks obtained by each student.

(b) The highest marks in each subject and the Roll No. of the student who secured it.

(c) The student who obtained the highest total marks.

```
#include<stdio.h>
Void main ()
{
 Int i, j, n, m, rollno [4];
Float marks [50][4];
Printf ("number of students");
Scanf ("%d", &n);
For (i=0; i<n;i++)
{
Marks [3][4]=0;
Printf("\n enter three scores of students%d\n",i+1);
For(j=0; j<3; j++)
{
Scanf("%f", & marks[i] [j]);
Marks[i][3]=marks[i][3]+marks[i] [j];
}
//To find highest marks in each subject.
For(i=0; i<4; i++)
{
Marks[n][i]=marks[0] [i];
For( j=1;j<n;j++)
{
If(marks[n][i]<marks[ j] [i])
```

```
{
Marks[n][i]=marks[j] [i];
Roll no[i]=j;
}
}
}
For(i=0;i<n;i++)
{
Printf("total marks obtained by %d student is
%f\n",i+1,marks[i][3]);
}
For( i=0; i<3; i++)
{
Printf("the highest marks is %d subject is
%f\n", i+1,marks[n][i]);
Printf("rollno\n", rollno[j]);
}
Printf("the student %d secured highest marks are % f",
rollno[3],marks[n][3]);

Getch();

}
```

3. (a) Explain the way of defining, opening and closing a file. Also explain the different modes of operation.

File is a data structure provided by C to handle input or output to a program through disks.

Files are used to keep information safe for a long period.

They are organized by giving each file a unique name.

The operations on files include:

1. Opening the file (to put or get information).
2. Reading information from the file.
3. Writing information on to the file.
4. Closing the file.

In order to do all the above operations we need the following handling functions:

Function Name	Operation
fopen ()	to open an existing file or to create a new one.
fclose()	To close a file which is open.
Getch(), fscanf(), getw()	To read the data from the file.
Putch(), fprintf(), putw()	To write data to files.
Fseek()	To position the file pointer to a desired place in the file.
Ftell()	To give (tell) the current position of the file.
Rewind()	To position the file at the beginning.

FILE OPENING MODES AND THEIR MEANINGS:

File opening mode:	Meaning
"w"	create a file, if doesn't exist delete the contents, if the file already exists.
"a"	append (add) data to a file at the end, if the file does not exist, create the file.
"r"	read data from the file if it exists. An error message will be produced, if the file doesn't exist.
"r+"	Read and write on the files.
"w+"	Same as r+.
"a+"	Adds read permission to a file which was opened in the "a" for working.

(b) Write a C program to read data from the keyboard, write it to a file called INPUT, again read the same data from the INPUT file, and display it on the screen.

```
#include<stdio.h>
main()
{
Char ch;
FILE *fp;
clrscr();
Fp=fopen[("Input",''w");
If(fp==NULL)
{
      Printf("file is not opened");
       exit(0);
```

```
}
While((ch=getchar())!=EOF)
{
    putc(ch,fp);
}
fclose(fp);
fp=fopen("Input",‘‘r");
while((ch=getc(fp)!=EOF)
printf("%c",ch)
}
```

4. (a). Distinguish between an array of structures and array within a structure. Give an example of each.

Array of structures is nothing but many structures with the same fields. It is a collection of similar data types.

Grouping structure variables into one array is referred to as an array of structures.

Examples:

```
struct student
{
    int marks 1;
    int marks 2;
    int marks 3;
}st;
```

If we declare a structure such that the fields has an array type declaration then array within a structure is defined. It enables the user to have decreased number of variables.

Example:

```
struct student
{
    int marks[3];
}st;
```

Example for array of structures

```
struct student
{
    int rno;
    char name[30];
    float marks;
} s[10];
```

Here s[10] represents array of structures. It consists of 10 student structures. The elements in the structure can be accessed as

```
s[i]. rno

s[i].name

s[i].marks
```

Array index starts from 0 and ends at size–1.

(b). Write a C program using structure to create a library catalogue with the following fields: Access number, Author's name, Title of book, Year of publication, Publisher's name, Price.

```
#include<stdio.h>
Struct library
{
Int acno;
Char aname[20];
Char tbook[40];
Int year;
Char pname[40];
Float price;
};
main()
{
Struct library s[50];
Int n;
Printf("enter the number of entries");
Scanf("%d",&n);
Printf("enter the book details");
For(I=0; I<=n; i++)
{
Scanf("%d%s%s%d%s%f",&s[i].acno,s[i].aname,s[i].tboo
k,&s[i].year,&s[i].pname,s[i].price);
}
Printf("\n Library catalogue\n");
Printf("Access number\t author number\t title of book\t
```

```
year of put\t pub name \t price \n");
For(i=1; i<=n; i++)
{
Printf("%d\t%s\t%s\t%d\t%s\t%f\n",s[i].acno,s[i].aname,s[i].
tbook,s[i].year,s[i].pname,s[i].price);
}
}
```

5. Write a C program to read information about the student record containing student's name, student's age and student's total marks. Write the marks of each student in an output file.

```
#include<stdio.h>
main()
{
FILE *f,*p;
Char sname[20];
Int age,n;
Float tot;
Printf("enter the number of students");
Scanf("%d",&n);
F=fopen("record",''w");
For(i=0; i<=n; i++)
{
Printf("enter student"s name, age and total");
Scanf("%s%d%f",&sname,&age,&total);
Fprintf(f,"%s%d%f",sname,&age,&tot);
}
Fclose(f);
F=fopen("record","r");
P=fopen("total",''r");
For(i=0; i<n;i++)
{
Fscanf(f,"%f",&tot);
Fprintf(p,"%f",tot);
}
```

```
Fclose(f);
Fclose(p);
P=fopen("total","r");
For(i=0;i<=n;i++)
{
Fscanf(p,"%f",& tot);
Printf("% f",tot);
}
}
```

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

The operations that can be performed on circular queue are enq(), deq(), traverse().

```
#define size 3
Int rear=–1,front=0, n=0, a[size];
Enq(e)
Int e;
{
If((rear+1)%size==front&&n=size)
{
Printf("overflow\n");
}
Else
{
Rear=rear+1;
A[rear]=e;
N++;
}

}

Deq()

{

Int x;

If((rear+1)%size=front&&n==0)

{
```

```
Printf("underflow\n")';
}
Else
{
X=a[front];
Front++;
N−−;
Return(x);
}
}
Traverse()
{
Int I;
If(n!=0)
{
printf("elements of circular queue:");
for(i=front; i!=rear; i=(i+1)% size)
{
Printf("%d",a[i]);
}
else
printf("empty\n");
main()
{
Int op,x,v;
Do
Printf("menu");
Printf("1.enq");
Printf("2.deq");
Printf("3.traverse");
Printf("4.exit);
Printf("enter your choice");
Scanf("%d,&op);
Switch(op)
{
```

```
Case 1:  printf("enter data element");
         Scanf("%d",&x);
         Enq(x);
         Break;
Case 2:  v=deq();
         If(v!=0)
         {
           Printf("deleted item=%d",v);
         }
         Else
         Printf("empty\n");
         Break;
Case 3:  Traverse();
         Break;
Case 4:  Exit(0);
}
While(1);
Getch();
}
```

7. Circular linked lists are usually setup with so called list header. What is the reason for introducing such a header? Write functions to insert and delete elements for this implementation.

A header in a linked list serves as the starting point to begin traversing the nodes of the list. In case of circular list, the last node is linked back to the first node, thus forming a loop. In this case, if we begin traversing from the first node, after we search the end node, we will again come back to the first node. However, in a program there is no identification, which is the first node, which is the last node, and so on. Thus, our program will go in an indefinite loop. To prevent this, a header is introduced in such lists. The header will point to the first node of the list.With this, we can know when we have to stop traversing.

```
    Struct linkedlist *get_node()
{
Struct linkedlist *tempnode;
Tempnode=(struct linkedlist *)malloc(size of(struct
linkedlist));
If(tempnode==NULL)
```

```
{
Printf("\n memory allocation failed");
Exit(1);
}
Return tempnode;
}
Struct linkedlist *insert_node(struct linkedlist *mynode, int
element)
{
Struct linkedlist *myheadernode,
*tempnode,*thisnode,*lastnode;
Myheadernode=mynode;
Tempnode=get_node();
Tempnode→ data=element;
If(mynode==NULL)
{
Myheadernode=tempnode;
Tempnode→next=myheadernode;
 }
Else
{
Last node=thisnode=myheadernode;
While(thisnode→data<=element&&
thisnode→next!=myheadernode)
{
Last node=thisnode;
Thisnode=thisnode→next;
}
If(thisnode→next==myhedernode && thisnode→data
<=element)
{
Tmpnode→next=thisnode→next;thisnode→next=tempnode;
}
Else if(thisnode!=myheadernode)
{
Tempnode→next=thisnode;
Lastnode→next=tempnode;
}
Else
```

```
{
While(thisnode→next!=myheadernode)
Thisnode=thisnode→neat)
Tempnode→next=lastnode;
Thisnode→next=tempnode;
Return tempnode;
}
}
 return myheadernode;
}
Struct linkedlist *delete_node(struct linkedlist *mynode, int
element)
{
Struct linkedlist *myheadernode, *tempnode, *thisnode,
*lastnode;
Myheadernode=thisnode=lastnode=mynode)
{
Lastnode=thisnode;
Thisnode=thisnode→next;
}
If(lastnode==mynode && lastnode→data==element)
{
While(thisnode→next!=mynode)
Thisnode=thisnode→next;
Thisnode→next=lastnode→next;
Tempnode=lastnode→next;
Free(lastnode);
Return tempnode;
}
If(thisnode→data==element)
{
Tempnode=thisnode;
Lastnode→next= thisnode→next;
Free(tempnode);
Return myheadernode;
}
Printf("\n no such element");
Return my headernode;
}
```

8. (a) Explain the algorithm for exchange sort with a suitable example.

We can also call exchange sort as bubble sort.

Algorithm Bubble (Arr, n)

Arr is an array of *n* elements

1. Repeat for $i = 0, 1, 2, 3, \ldots\ldots n-1$.
2. Repeat for j = i+1 to $n-1$.
3. if(Arr[i]>Arr[j]) then interchange Arr[i] and Arr[j] end if
4. increment j by 1.
5. end for
6. end for
7. print the sorted array Arr end bubble.

The idea of bubble sort is to repeat by moving the smallest element to the lowest index position in the list. The process is clearly explained in the above algorithm.

Example:-

15 18 3 9 12

The first pass takes $i = 1$ and $j = 2, 3, 4, 5$. The value 15 is compared with 18 and net changed 15>3, now values interchanged 3, 18,15, 9, 12.

Since 3<9 and 3<12, so elements are not changed

After the first pass, list is 3, 18, 15, 9 ,12

In the second pass $i = 2$ and $j = 3, 4, 5$

Values 18>15, so interchanged 3, 15, 18, 9,12

Since 15>9, so interchanged 3, 9, 18, 15, 12

Since 9<12, no change is required 3, 9, 18, 15, 12

In the third pass $i = 3$ and $j = 4, 5$.

Since 18>15, so interchanged 3, 9, 15, 18, 12

Since 15>12, so interchanged 3, 9, 12, 18, 15

In the fourth pass, $i = 4$ and $j = 5$

Since 18>15, so interchanged 3, 9, 12, 15, 18

Efficiency of bubble sort:

No. of comparison in first pass = $n-1$

No. of comparisons in second pass = n-2 and so on.

The total no. of comparisons

$(n\text{-}1)+(n\text{-}2)+(n\text{-}3)+\ldots\ldots\ldots\ldots+2+1=n(n+1)/2 = O(n2)$

(b) Compare sort and exchange sort.

Sorting refers to the operation of arranging records in some given order. Sorting can either be internal or external. When sorting is done keeping the records in the main memory, it is called internal sorting. When sorting is done by keeping the records in external files on storage devices like disks, tapes, etc., it is called external sorting.

We have different sorting algorithms to sort the data They are:

Bubble sort(or) Exchange sort

Insertion sort
Selection sort
Quick sort
Heap sort
} Internal sorts

Merge sort ← External sorts

Sorting time summary

	Work case	Average case
Bubble sort	O(*n*^2)	O(*n*^2)
Quick sort	O(*n* ^ 2)	O(*n* log *n*)
Insertion sort	O(*n*^2)	O(*n*^2)
Selection sort	O(*n*^2)	O(*n*^2)
Merge sort	O(*n* log *n*)	O(*n* log *n*)
Heap sort	O(*n* log *n*)	O(*n* log *n*)

SELECTING A SORT:-

Bubble sort	Good for small *n*(*n*<=100)
Quick sort	Excellent for virtual memory environment
Insertion sort	Good for almost sorted data
Selection sort	Good for partially stored data and small *n*
Merge sort	Good for external file sorting
Heap sort	As efficient as quick sort in an average case and far superior to quick sort in work case.