Exam -1 Assignment

C programming

1. What is recursion? Write a factorial program in C using recursion.
2. What is the purpose of '\0' character in C;
3. Write a program to find whether the given number is an Armstrong number or not?
4. 1*3 + 2*5 + 3*7 + …. + n*(2n+1)
5. Difference between call by value and call by reference with example. What is the difference between while and do while loop?
6. Input a String an print them Reversely.

Data Structure

1. Difference between Divide and Conquer and Dynamic Programming
2. There are no well-defined standards for writing algorithms. Efficiency of an algorithm depends on several factors. Similarly, complexity of an algorithm also depends of several factors. Describe the algorithm complexity factors.
3. Inorder Traversal : { 4, 2, 1, 7, 5, 8, 3, 6 }, Preorder Traversal: { 1, 2, 4, 3, 5, 7, 8, 6 } construct tree.
4. Draw a Binary min heap that results from inserting 9, 3, 7, 4, 8, 2, 6, 5 in that order into an initially empty heap. Show each insertion step separately. You do not need to show the array representation. Just draw the trees.
5. Write down time complexities of different sorting algorithm.
6. Draw a flowchart to input five positive numbers, and sort them is ascending order.
7. Explain BFS Traversal of a graph with an example

8. Assume we have an array storing the values 3, 9, 12, 13, 24, 25, 26, 200, 202, 208, 215, 300, 314, 31415, 80000. The lowest value is stored at index 0 and the highest at index 14.

i. Suppose we want to check whether 14 are in the array or not. Which values does a binary search ended to compare with?
ii. How many comparisons would you need if you did a linear search and list out compared values?
iii. Different binary search and linear search comparison using time complexity equation where assume that we have an array of size 10,000.

What is an algorithm? Explain with its characteristics and example.

- Define data structure. Classify data structure.
- Explain data structure operations and list the importance of data structures.
- Explain an Abstract Data Type.
- Discuss on time and space complexity of algorithms with examples.
- Define the worst case, best case and average case complexities with examples.

- Write short notes on asymptotic notations.
- Explain Stack data structure with its applications.
- Write down the algorithms for push and pop operations on stack.
- Explain stack as an ADT.
- Write down the algorithms for converting
- Infix expression to postfix expression. b) Infix expression to prefix expression.
- Write down the algorithms for evaluating:
- Postfix expression b) Infix expression c) Prefix expression6.
- Translate the following infix expression into its equivalent postfix expression using algorithm:((A-(B+C))*D)$(E+F)7.
- Evaluate the following postfix expression:AB+C*DEFG*+/+H-, Where A=4, B=8, C=2, D=5, E=6, F=9, G=1, H=3
- Explain queue data structure with its applications.
- Explain queue as an ADT.
- Differentiate between linear and circular queue
- Write down the algorithms for enqueue and dequeue operation in a linear queue.
- Write down the algorithms for enqueue and dequeue operation in a circular queue.
- Explain the priority queue with its operations and applications
- Given are two one-dimensional arrays A and B which are stored in ascending order.Write a program to merge them into a single sorted array C that contains every item fromarrays A and B, in ascending order.
- Twentyfive numbers are entered from the keyboard into an array. Thenumber to be searched is entered through the keyboard by the user. WAP tofind if the number to be searched is present in the array and if it is present,display the number of times it appears in the array.

C Output Problem:

C Programming Output Problems

Cloud IT Online- Class -5

STUDY MATERIAL: Cloud IT Online (Handbook: C output Problem, Practice Book: C , Exercise Book: C Programming Problem chapter- 1, Lecture Sheet : For C Programming Theory)

Previous Year Output Problems: Cloud IT Online: Handbook: C Output Problems

1. How can we store 5.95 as a float?

A. use 5.95f
B. use 5.95
C. use 5.95ld
D. none of the above.

Output

```
A) use 5.95f
```

Explanation

In case of C and C++ programming 5.95 treat as double value by default. To convert double value into float value we have to use ‘f’ at the end of value. To specify 5.95 as long double we have to use character ‘L’ at the end of value.

```
#include <stdio.h>

int main()
{
		char ch='A';
		printf("%d",ch);
		return 0;
}
```

Output

```
65
```

Explanation

Here, value of **"ch"** is **'A'** and we are printing the value **"ch"** in integer format (using **"%d"** format specifier). Thus, the output will be **65** (which is the **ASCII code uppercase character 'A'**).

```
#include <stdio.h>

int main()
{
		char ch='A';
		printf("%c-%d",ch+32,ch+32);
		return 0;
}
```

Output

```
a-97
```

Explanation

Variable contains **ASCII code of 'A' which is 65**. And we are printing **"ch+32"** which will be **97** and the format specifier **"%c"** will print **"a"** and **"%d"** will print **97**.

```
int main()
{
		int x[10]={0,1,2,3,4,5,6,7,8,9};
		int *ptr1,*ptr2;
		ptr1=&x[0];
		ptr2=&x[5];
		printf("%p\n",(ptr1+ptr2));
		return 0;
}
```

Output

```
error: invalid operands to binary + (have 'int *' and 'int *')
```

Explanation

Error in the statement **printf("%p\n",(ptr1+ptr2));**
Addition of two pointers is not allowed.

```
int main()
{
		int color=2;
		switch(color)
		{
				case 0: printf("Black");
				case 1: printf("Blue");
				case 2: printf("Green");
				case 3: printf("Aqua");
				default: printf("Other");
		}
		return 0;
}
```

Output

```
GreenAquaOther
```

Explanation

There are no **break** statements, so all the statements after **case 2** will be executed including **default** statement.

```
int main()
{
		if(0);
		printf("Hello");
		printf("Hi");
		return 0;
}
```

Output

```
HelloHi
```

Explanation

There is a semicolon after the **if** statement, so this statement will be considered as separate statement; and here **printf("Hello");** will not be associated with the **if** statement. Both **printf** statements will be executed.

```
#include <stdio.h>

int main()
{
   int i;
   for(i=65; i<(65+26);i++)
       printf("%c ",i);
           return 0;
}
```

Output

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

Explanation

See the **printf** statement, here we are printing the value in character format (using **"%c"** format specifier) and the value of **i** is 65 to 90, which are the **ASCII codes of Uppercase alphabets from A to Z**. Thus, the output will be **"A B C ... Z"**.

What will happen if we assigned a value to an array element whose size of subscript is greater than the size of array in C programming?

A. A compilation error occurs.
B. The element will be set 0.
C. The program crashes at run time.
D. The array size increases automatically.

Output

```
C) The program crashes at run time.
```

Explanation

In C programming if we assign a value to array element whose size of subscript is greater than size of subscript of array then it crashes at run time because array have fixed size for data management. We cannot extend or reduce the size of array. Due to this drawback of array, linked list is formed. In linked list we can increase or decrease the size. But now modern gcc compilers will take care of this kind of errors.

```
int main()
{
		int x,y;
		int *ptr;
		x=100;
		ptr=&x;
		y=*ptr;
		printf("%d\n",y);
		return 0;
}
```

Output

```
100
```

Explanation

Here, **y** is assigned the value stored at the pointee variable, which is pointer through **ptr**.

```
#include <stdio.h>
#include <string.h>
int main()
{
		char str[20] = "ABCDEFGHIJK";
		int s = strlen(str);

		str[3] = '\0';
		s = strlen(str);

		printf("%d\n",s);
		return 0;
}
```

Output

```
3
```

```
#include <stdio.h>
void main()
{
```

```
            int a=10,b=2,x=0;
            x=a+b*a+10/2*a;
            printf("value is =%d",x);
}
```

Output: **value is =80**

```
#include <stdio.h>
void main()
{
            int x;
            x= (printf("AA")||printf("BB"));
            printf("%d",x);
            printf("\n");

            x= (printf("AA")&&printf("BB"));
            printf("%d",x);
}
```

Output: AA1
AABB1

```
#include <stdio.h>
int main()
{
            int i=-1,j=-1,k=0,l=2,m;
            m=i++&&j++&&k++||l++;
            printf("%d %d %d %d %d",i,j,k,l,m);
            return 0;
}
```

Output:
0 0 1 3 1
inti=-1,j=-1,k=0,l=2,m;
m=i++&&j++&&k++||l++;

++ Operator has precedence over &&, || operator,
thus it is evaluated first. So the expression turns to be
m= 0 && 0 && 1 || 3
i=0. j=0, k=1, l=3
Now && has precedence over || and both has associativity
left to right. Hence the expression part 0 && 0 is evaluated first, so on.
m=((0 && 0) && 1) || 3)
 =((0 && 1) || 3)
= 1 || 3= 1
Thus value of
m=1
i=0
j=0
k=1
l=3