1. Before putting a value on the stack, we must use Top and Max to make sure the stack is full or empty. If a value is inserted while the stack is full, an overflow occurs. If a value is inserted while completely empty, an underflow occurs.
2. The basic operations of arrays are insert, delete, modify, search, and sort. The basic operations of the stack are push, pop, and peek. Elements of an array can be accessed through an index, but elements on the stack can only be read or deleted at a time from the topmost element.
3. When implementing a stack using an array, it is possible if the stack size is small or predetermined. However, if the stack size is large or unspecified, it must be implemented using a linked list. This is because the linked list can continuously create nodes without limitation through dynamic memory allocation.
4. pop () is used to remove the topmost element from the stack, and peek () is used to return the topmost element.
5. Because the operator which occurs first in the expression is operated first one the operands. For example, given a postfix notation AC-B*. while evaluation, subtract will be performed prior to multiplication.
6. The same way they are used in a program with recursive functions.
7. It means that one stack is divided into two spaces. The first stack is stacked from left to right, and the second stack is stacked from right to left.
8. Infix notation means there is an operator between operands, and it is usually the form we use in algebra. But it is easy for human to write expressions using infix notation, computers find it difficult to parse as the computer needs a lot of information to evaluate the expressions(operator precedence, associativity rules, brackets which override these rule). Thus, prefix notation and postfix notation, which are efficient ways for computers to operate, were created. In postfix notation, the operator is placed after the operands. In prefix notation, the operators are applied to the operands that are present immediately on the right of the operator.
   (a) AB-C+
   (b) AB*CD/+
   (c) AB-CD*E/+C-
   (d) AB*CD/+DE+-
   (e) AB-DEF+G*/+
   (f) A2BC+*D/E*-F+
   (g) 14 7/ 3* 4-9 2/+
9. infix expressions -> prefix expressions
   (a) +-ABC
   (b) +*AB/CD
   (c) -+-AB/*CDEC

(d) -+*AB/CD+DE

(e) +-AB/D*+EFG

(f) +-A*/*2+BCDEF

(g) +-*/14 7 3 4 /92

10.

(a) (A+B)*C-D

(b) A+B*C-D

11.

(a) ((a+b)-c)*d

(b) a-B*C+D

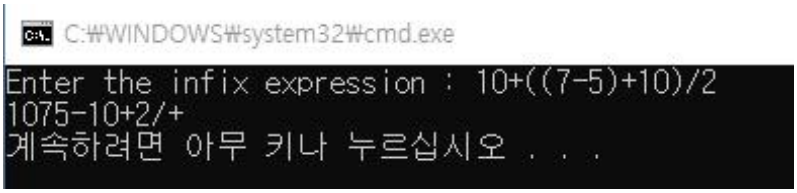
12.

<infix->postfix expression>

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>
# define max 500
char stack[max];
int top=-1;
void change(char input[], char output[]);
void push(char input[], char value);
char pop(char input[]);
int prior(char value);
int main()
{
        char input[500],output[500];
        int i;
        printf("Enter the infix expression : ");
        gets(input);
        strcpy(output, "");
        change(input,output);
        puts(output);

}


void change(char input[], char output[])
{
        int i=0,j=0;
        char temp;
        while(input[i]!='\0')
        {
```

```
		if(input[i]=='(')
		{
			push(stack,input[i]);
			i++;
		}
		else if(input[i]==')')
		{
			while((top!=-1) && (stack[top]!='('))
			{
				output[j]=pop(stack);
				j++;
			}
			if(top==-1)
			{
				printf("incorrect\n");
			}
			temp=pop(stack);
			i++;
		}
		else if(isdigit(input[i]) || isalpha(input[i]))
		{
			output[j]=input[i];
			i++;
			j++;
		}
		else if(input[i] == '+' || input[i] == '-' || input[i] == '*' || input[i] == '/' ||
input[i] == '%')
		{
			while((top!=-1) && (stack[top]!='(') &&
(prior(stack[top])>=prior(input[i])))
			{
				output[j]=pop(stack);
				j++;
			}
			push(stack,input[i]);
			i++;
		}
		else
		{
			printf("incorrect element in expression\n");
			exit(1);
		}
	}
```

```
        while((top!=-1) && (stack[top]!='('))
        {
                output[j]=pop(stack);
                j++;
        }
        output[j]='\0';
}

void push(char input[], char value)
{
        if(top==max-1)
                printf("Stack overflow\n");
        else
        {
                top++;
                stack[top]=value;
        }
}

char pop(char input[])
{
        char value=' ';//top==-1일때 초기화 해놓지 않으면 오류발생
        if(top==-1)
                printf("stack underflow");
        else
        {
                value=stack[top];
                top--;
        }
        return value;
}

int prior(char value)
{
        if(value=='/' || value=='*')
                return 3;
        if(value=='%')
                return 2;
        if(value=='+' || value=='-')
                return 1;
}
```




13.

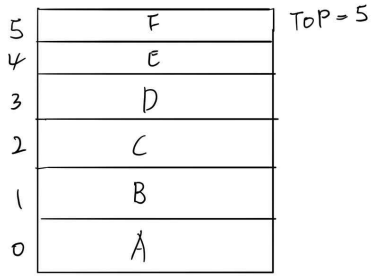
```
#include <stdio.h>
# define max 5
int stack_a[max];
int stack_b[max];
int top_a=-1;
int top_b=-1;
void copy(int stack_a[],int stack_b[]);
void push_a(int stack_a[], int val);
void push_b(int stack_b[], int val);
void display(int stack_b[]);
int main ()
{
        int i,option,val;
        do{
        printf("\n\n");
        printf("1.a_push\n");
        printf("2.b_puhs\n");
        printf("3.copy\n");
        printf("4.display\n");
        printf("5.exit\n");
        printf("Enter the option : ");
        scanf("%d",&option);
        switch(option)
        {
        case 1: printf("Enter the data : ");
                scanf("%d",&val);
                push_a(stack_a,val);
                break;
        case 2: printf("Enter the data : ");
                scanf("%d",&val);
                push_b(stack_b,val);
                break;
        case 3: copy(stack_a,stack_b);
                break;
        case 4: display(stack_b);
        }
        }while(option!=5);
```

```
}

void copy(int stack_a[],int stack_b[])
{
        int i;
        for(i=top_a;i>=0;i--)
        stack_b[i]=stack_a[i];
}
void push_a(int stack_a[], int val)
{
        if(max==top_a-1)
                printf("stack overflow");
        else
        {
                top_a++;
                stack_a[top_a]=val;
        }
}
void push_b(int stack_b[], int val)
{
        if(max==top_b-1)
                printf("stack overflow");
        else
        {
                top_b++;
                stack_b[top_b]=val;
        }
}

void display(int stack_b[])
{
        int i;
        if(top_b==-1)
                printf("stack underflow");
        else
        {
                for(i=top_b;i>=0;i--)
                        printf("%d",stack_b[i]);
        }

}
```

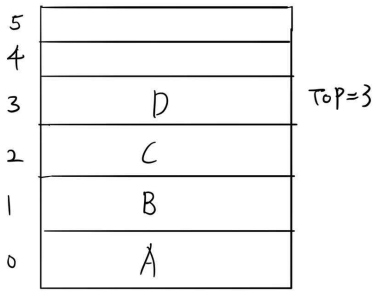
```
1.a_push
2.b_puhs
3.copy
4.display
5.exit
Enter the option : 1
Enter the data : 1


1.a_push
2.b_puhs
3.copy
4.display
5.exit
Enter the option : 1
Enter the data : 2


1.a_push
2.b_puhs
3.copy
4.display
5.exit
Enter the option : 1
Enter the data : 3


1.a_push
2.b_puhs
3.copy
4.display
5.exit
Enter the option : 2
Enter the data : 4


1.a_push
2.b_puhs
3.copy
4.display
5.exit
Enter the option : 2
Enter the data : 5


1.a_push
2.b_puhs
3.copy
4.display
5.exit
Enter the option : 2
Enter the data : 6


1.a_push
2.b_puhs
3.copy
4.display
5.exit
Enter the option : 3


1.a_push
2.b_puhs
3.copy
4.display
5.exit
Enter the option : 4
321

1.a_push
2.b_puhs
3.copy
4.display
5.exit
Enter the option : 5
계속하려면 아무 키나 누르십시오 . . .
```

14.

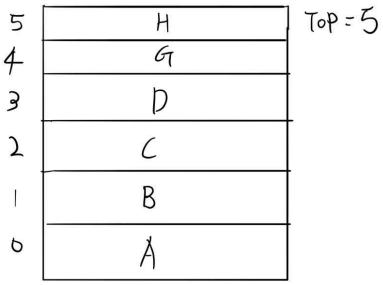
(a).



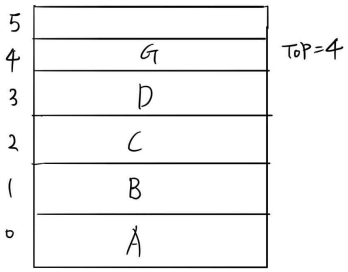

(b).



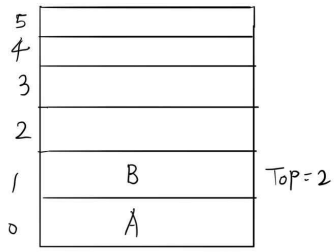

(c).



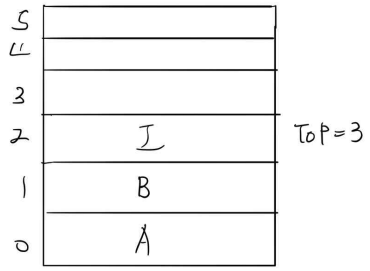

(d).




(e).




(f).




15. Recursion is an excellent way of solving complex problems especially when the problem can be defined in recursive terms. For such problems, a recursive code can be written and modified in a much simpler and clearer manner. But this solutions may require substantial amount of run-time overhead. Therefore, when implementing a recursive solution, there is a trade-off.

16. The Tower of Hanoi, is a mathematical problem which consists of three rods and multiple disks. Initially, all the disks are placed on one rod, one over the other in ascending order of size similar to a cone-shaped tower.

The objective of this problem is to move the stack of disks from the initial rod to another rod, following these rules:
1)A disk cannot be placed on top of a smaller disk
2)No disk can be placed on top of the smaller disk.

Multiple-choice Questions
1. (a)
2. (b)
3. (a)
4. (c)

True or False
1. F
2. T
3. T
4. T
5. F
6. T
7. F
8. T
9. F
10. T
11. F
12. F

Fill in the blanks
1. stack
2. stack
3. O(n)
4. top=NULL
5. left to right
6. non-tail
7. direct