11-7-17

UNIT-II

Stacks & Queues

Templates:-

* A template is a method for creating a single function (or) class for a family of similar type functions (or) classes for making (or) creating generic program
* Basically templates are classified into two types they are
  i) function template
  ii) class template

function template:-

If a function is created for a group of similar type functions using the templates then such function can be called as function template

Syntax:-

```
template <class T>
T function_name(T argument)
{
  :
  :
  return(T);
}
```

* where template, class are the keywords in c++ and T is a generic data type (or) parameterised data type
* the function template must contain atleast one generic data type argument.

Ex:-

```
#include<iostream.h>
#include<conio.h>
template <class T>
T getmax (T a,T b)
  {
T result;
result=(a>b) ? a:b;
```

```
return result;
}
int main()
{
 int ix, iy;
 float fx, fy;
 clrscr();
 cout<<"enter any two integer values:"<<endl;
 cin>>ix>>iy;
 cout<<"Enter any two floating point values:"<<endl;
 cin>>fx>>fy;
 int k=getmax(ix,iy);
 float n=getmax(fx,fy);
 cout<<"the max value in integers is:"<<k<<endl;
 cout<<"the max value in float values is:"<<n<<endl;
 getch();
 return 0;
}
```

```
enter any two integers values:
8
enter any two floating point values
9.5  5.8
The max value in integers is: 8
The max value in float is: 9.5
```

Class Template:

* If a class is created for a group of similar classes using template keyword then such class can be called as class template
* The class template can work on different data types without rewritting specific class for each data type

Syntax:-

```
template <class T>
 class class_name
 {
 private:
   :
 public:
   :
```

Ex:-

```
# include <iostream.h>
# include <conio.h>
  template <class T>
  class Sample
   {
   private:
    T a, b;
   public:
    void getdata( );
    void Sum( );
   };
   template <class T>
   void Sample <T>::getdata( )
   {
    cin >> a >> b;
   }
   template <class T>
   void Sample <T>::Sum( )
   {
   T c;
   c = a+b;
   cout << "Sum is=" << c << endl;
   }
  void main( )
   {
   Sample <int> obj1;
   Sample <float> obj2;
   clrscr( );
   cout<<"Enter any two integer values:"<<endl;
   obj1.getdata( );
   obj1.Sum( );
   cout<<"Enter any two floating point values:"<<endl;
   obj2.getdata( );
   obj2.Sum( );
   getch( );
   }
```

O/p:-

```
Enter any two integer values:
5   8
Sum is=13
Enter any two floating point values:
3.5   8.5
Sum is=12
```

Note:- The templates which may be either function template (or) class template do you work on different data types to perform different operations without re-writting specific function (or) class for each data type

Advantage:- The templates can reduce the length of the program code by writting once and used by many different data types

Stack ADT:- → It must be elements and along with operations

* A Stack is an abstract data type because it contains (or) stores set of elements and has some associated operations
* A stack is a linear data structure which stores the elements in a sequencial order.
* Usually the stack follows a principle known as LIFO (last-in-first-out) which mean that the element which is inserted last into the stack will be deleted first

There are two operations will be performed on the stack they are 1. push
2. pop

Push:- The push operation is used to insert a new element into the stack.

Pop:- The pop operation is used to delete an element from the stack.

* In general the stacks uses a special small register which is know as stacks pointer register which will be refered as top and whose initial values -1 → to store the address of the stack
* The stack pointer register value can be incremented (or) decremented to perform either push (or) pop operation on the stack

* while performing push operation on the stack, the stack pointer register value must be incremented and then the new element will be inserted into the stack.

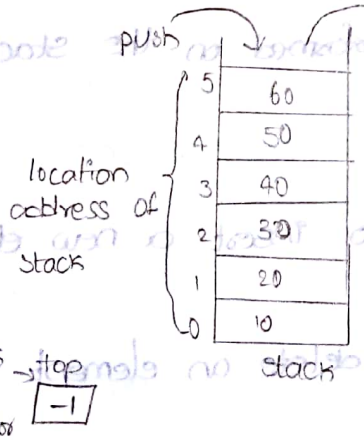
* while performing pop operation on the stack, first the top element must be deleted from the stacks and then the stack pointer register value should be decremented by 1.

* On the stack, the push and pop operations can be performed from only a single end which is known as end of the stack.

* When we try to insert an element into the stack, even though the stacks is full of elements. then a situtation is occured known as stack overflowed

* When we try to delete an element from the stack, even though the stack is empty then a situation is occured know as stack under flow

Ex:- 10, 20, 30, 40, 50, 60



* The stack can be implemented using either arrays (or) linked lists

Ex:-
```
#include <iostream.h>
#include <conio.h>
int stack[100], n, choice, top, x, i;
void push();
void pop();
```

(length of array; to indicate the top; we use switch operation; global declaration)

```
void display();
void peek();
void main()
{
 top=-1;
 clrscr();
 cout<<"Enter the size of the stack"<<endl;
 cin>>n;
 cout<<"the stack operations are:"<<endl;
 cout<<"1.push"<<"\t"<<"2.pop"<<"\t"<<"3.display"<<"\t"<<"
                                    4.peek"<<"\t"<<"5.Exit"<<endl;
 do
  {
  cout<<"Enter your choice"<<endl;
  cin>>choice;
  switch(choice)
  {
   case 1:
    {
     push();
     break;
    }
   case 2:
     {
      pop();
      break;
     }
   case 3:
     {
      display();
      break;
     }
   case 4:
     {
      peek();
      break;
     }
   case 5:
   {
   cout<<"Exit"<<endl;
   break;
  }
```

```
default:
 {
 cout<<" Enter a valid choice (1/2/3/4/5):" <<endl;
  }
 }
}
while (choice!=5);
 getch( );
}
 void push( )
 {
 if (top >=n-1)
{
 cout<<" stack is overfloa"<<endl;
 }
else
  {
 cout<<"the element to be inserted into the stack is:";
  cin>>x;
  top++;
  stack [top]=x;
 }
}
 void pop( )
 {
 if (top<=-1)
  {
 cout<<" stack is underflow"<<endl;
  }
 else
  {
 cout<<"the deleted element is:"<<stack [top];
   top--;
  }
 }
 void display( )
 {
 if (top>=0)
```

```
cout<<" the elements in the stack are:"<<endl;
for(i=top; i>=0; i--)
 {
  cout<<Stack[i]<<endl;
 }
 cout<<"enter the next choice:"<<endl;
 }
else
 {
 cout<<" the stack is empty: "<<endl;
 }
}

void peek( )
 {
if(top==-1)
 {
 cout<<" the stack is empty"<<endl;
 }
else
 {
 cout<<" the top element in the stack is:"<< Stack[top]<<endl;
 }
}
```

Queue ADT:-

* A queue is also an abstract data type why because it contains set of elements (or) storage locations and some associated operations

queue

* A queue is a linear data structure which stores the elements in a sequencial order

* Usually the queue follows a principle known as FIFO (first in first out) which mean that what ever the element which is inserted first into the queue will be deleted first.

* On the queue we perform two basic operations they are
  1) Enqueue
  2) Dequeue

ENQUEUE:-

* The enqueue operation is used to insert a new element into the queue

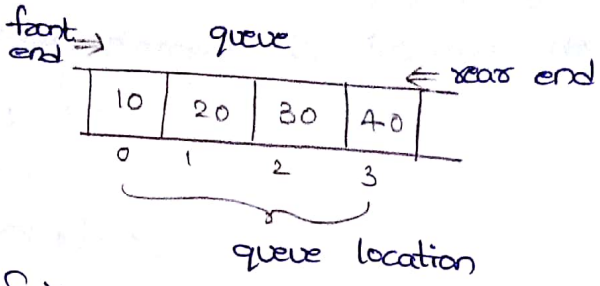
Dequeue:-

* The dequeue operation is used to delete an element from the queue

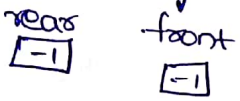
* The queue uses two small registors known as rear and front which are used to hold the address of queue location while performing enqueue and dequeue operations. The rear and front registors initial values are assigned to -1.

* In order to perform enqueue operation, first we must increment the rear registor value by 1 and then the new element will be inserted into the queue

* In order to perform dequeue operation, first we must decrement the front registor value by 1 and then the element will be deleted into the queue.



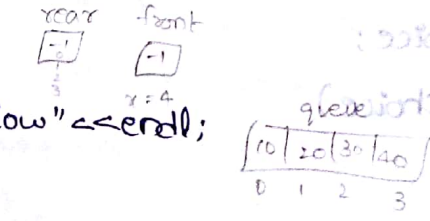
Initially



program for the queue implementation using arrays.

```
#include<iostream.h>
#include<conio.h>
int queue[100], n, rear, front, choice, x, i;
void enqueue( );
void dequeue( );
void display( );
void main( )
{
rear = -1;
front = -1;
clrscr( );
cout<<"enter the size of the queue:";
cin>>n;
cout<<" the queue operations are:"<<endl;
cout<<"1. enqueue"<<"\t"<<"2. dequeue"<<"\t"<<"3. display"<<
                "\t"<<"4. exit"<<endl;
do
{
cout<<"enter your choice:";
cin>>choice;
switch (choice)
{
case 1:
{
```

```
enqueue();
break;
}
case 2:
{
dequeue();
break;
}
case 3:
{
display();
break;
}
case 4:
{
cout<<"exit"<<endl;
break;
}
default:
{
cout<<"enter a valid choice (1/2/3/4):"<<endl;
break;
}
}
}
while(choice!=4);
getch();
}
void enqueue
{
if(rear>=n-1)
{
cout<<"queue is overflow"<<endl;
}
else
{
```



```
cout<<"the element is inserted into the queue is:";
cin>>x;
rear++;
queue[rear]=x;
}
}
void dequeue()
{
if (front==rear)
{
cout<<" the queue is underflow"<<endl;
}
else
{
front++;
cout<<" the deleted element from the queue is:"<<queue[front]<<endl;
}
}
void display()
{
if (front==rear)
{
cout<<" the queue is empty"<<endl;
}
else
{
cout<<"the list of elements stored in the queue are:";
for (i=front+1; i<=rear; i++)
{
cout<<queue[i]<<" ";
}
}
}
```

Expression:-

An expression is the combination of operands and operators which ultimately represents a value.

Types of expression:

Based on the position of the operator the expressions are classified into 3 types they are 1. Infix expression
2. pre-fix expression
3. post-fix expression

Infix expression :-

In an expression if the operator is to be based b/w the operands then such expression can be called as infix expression.

General form:

operand1 operator & operand 2

Ex:- a+b

prefix expression:

In an expression if the operator is to be based b/w the first of the operands then such expression can be called as prefix expression.

GF

operator operand 1 operand 2

Eg:- +ab

postfix expression:-

In an expression if the operator is to be placed in the ending of the position of the operand then such expression can be called as postfix.

GF operand1 operand2 operator

Eg: ab+

we can convert one expression form into another expression like

Infix to prefix

Infix to postfix

prefix to postfix and vice-versa

** conversion from infix expression into postfix expression:-

Algorithm steps:-

Step 1: Start

Step 2: Add left parantheses "(" to the stacks and right parantheses ")" to the end of the given infix expression.

Step 3: Scan (or) read all the characters one by one from left to right in the given infix expression.

Step 4: If the character is an operand then place it into postfix expression.

Step 5: If the character is left parantheses "(" then push it into the stack.

Step 6: If the character is right parantheses ")" then pop the operator characters from the stack and place into postfix expression until respective left parantheses "(" is encountered in the stack.

Step 7: If the character is an operator then push it into Stacks.

If the current operator priority is >= to the already existed operator in the stacks the push it into stacks.

otherwise pop the existed operator from the stacks and push the current operator into stacks. and placed popped operation into postfix expression

Step 8: Stop

Eg: Convert the infix expression (A+B) *(C+D) into postfix expression.

Infix expression (input)	Stack	postfix expression (output)
(	(	
(	((	
A	((	A
+	((+	A
B	((+	AB
)	(	AB+
*	(*	AB+
(	(*(	AB+
C	(*(	AB+C
+	(*(+	AB+C
D	(*(+	AB+CD
)	(*	AB+CD+
)		AB+CD+*

A+(B/C+(D%E*F)/G)*H

A- (B/C + (D%E*F)/G) *H

high priority

/, *, %, ^

low

+, -

Infix	Stack	postfix
	(	
A	(	A
-	(-	A
(	(-(	A
B	(-(	AB
/	(-(/	AB
C	(-(/	ABC
+	(-(+	ABC/
(	(-(+(	ABC/
D	(-(+(	ABC/D
%	(-(+(%	ABC/D
E	(-(+(%	ABC/DE
*	(-(+(%*	ABC/DE
F	(-(+(%*	ABC/DEF
)	(-(+	ABC/DEF*%
/	(-(+/	ABC/DEF*%
G	(-(+/	ABC/DEF*%G
)	(-	ABC/DEF*%G/+
*	(-*	ABC/DEF*%G/+
H	(-*	ABC/DEF*%G/+H
)		ABC/DEF*%G/+H*-

x+(y*z/m)+s/(l/n%p)

infix	stack	postfix
	(	
x	(	x
+	(+	x
(	(+(	x
y	(+(	xy
*	(+(*	xy
z	(+(*	xyz
/	(+(/	xyz*
m	(+(/	xyz*m
)	(+	xyz*m/
+	(++	xyz*m/s
s	(++	xyz*m/s
/	(++/	xyz*m/s
(	(++/(	xyz*m/s
l	(++/(	xyz*m/sl
/	(++/(	xyz*m/sl
n	(++/(/	xyz*m/sln
%	(++/(/%	xyz*m/sln
p	(++/(/%	xyz*m/slnp
)	(++/	xyz*m/slnp%/
)		xyz*m/slnp%/++

Evaluation of post-fix expression :-

Algorithm steps :-

Step 1 :- Start

Step 2 :- Scan (or) read all the characters one by one from left to right in the given post-fix expression.

Step 3 :- If the characters is an operand then push it into the stack

Step 4 :- If the character is an operator then pop two operands from the stack and perform the operation with that operator from left to right and also push the result back into stack.

Step 5 :- Repeat Steps ③ & ④ until all the characters are scanned from the given post-fix expression

Step 6 :- pop the final result from the stack as output.

Step 7 :- Stop

Eg: Evalue the post-fix expression 23*45*+

post-fix expression	stack
	⊏
2	2 ⊏
3	2 3
*	6
4	6 4
5	6 4 5
*	6 20
+	26
	⊏
	∴ o/p :- 26

Evaluate the postfix expression 12+34+^

postfix	stack
	⊏
1	1
2	1 2
+	3
3	3 3
4	3 3 4
+	3 7
^	2187
	⊏
	o/p:- 2187

Subtyping:-

The process of creating sub class (or) sub types from their subclass (or) supertypes.

By creating sub class (or) subtypes we can achieve code reusability because the subtype can accesses (or) use all the properties from its super type.

Eg:-

```
#include<iostream.h>
#include<conio.h>
class A
{
public:
 int a,b
void getdata();
};
Class B:public A
{
```

```
int c,d;
public:
void add();
void sub();
void display();
};
void A::getdata()
{
cout<<"enter a value:"<<endl;
cin>>a;
cout<<"enter b value:"<<endl;
cin>>b;
}
void B::add()
{
c=a+b;
}
void B::sub()
{
d=a-b;
}
void B::display()
{
cout<<"addition is:"<<c<<endl;
cout<<"substraction is:"<<d;
}
int main()
{
B obj;
clrscr();
obj.getdata();
obj.add();
obj.sub();
obj.display();
return 0;
}
```