UNIT-V

Generic Programming with Templates

* What are Templates?
  - Templates are the important feature of C++ PL, that allows functions and classes to operate with generic data types.
  - Constructs a family of template functions or classes to work on many different data types without being rewritten for each one.
  - Templates can be
    → Function templates - templates declared for functions.
    → Class templates - templates declared for classes.

Function Templates

* Function templates can be used to create a family of functions with different argument types.

Syntax:

```
template <class T>                    → atleast one argument must
returntype fn_name (arguments)          be template type.
{
   // Body of function with type T
}
```

- The function template is prefixed with the keyword 'template' and the template parameter T (ie)

  template <class T>

- It tells the compiler that we are going to declare a template and use T as type name in declaration

- When the compiler encounters a call to template functions, it identifies the data type of parameters and creates a function internally and makes a call to it.
- Templates avoid overhead of rewriting functions having body of same pattern, but operates on different datatypes.

(Eg) Swap function

```
#include <iostream.h>
#include <conio.h>
template <class T>
void swap (T *a, T *b)
 {
  T t;
  t = a;
  a = b;
  b = t;
 }
void main()
 {
  char c1, c2;
  cout << "Enter two characters\n";
  cin>> c1 >> c2;
  swap(c1, c2); // invokes swap fn with char type var.
  cout << "After swapping "<< c1 << " "<< c2;

  int x, y;
  cout << "\n Enter two integers\n";
  cin>> x >>y;
  swap(x,y); // invokes swap fn with int type variable.
  cout << "After swapping "<< x << " " << y;

 }
```

O/P

```
Enter two characters
f y
After swapping
y f
Enter two integers
4 3
After swapping
3 4
```

* Here the compiler has created two swap fns. operating on char and int internally which is invisible to user.

Generic function for sorting algorithm (Bubble sort)

* Sorting is most commonly used data processing application.
* Sorting functions can be declared as function template & can be used to sort data items of any type.

(Eg) Bubble sort.

```
#include <iostream.h>
template <class T>
void swap (T &a, T &b)
 {
  T t;
  t = a;
  a = b;
  b = t;
 }
template < class bubble>
void bubblesort (bubble *a, int n)
 {
  for (int i=0 ; i<n-1 ; i++)
  for (int j=0 ; j<(n-1)-i ; j++)
   {
    if (a[j] > a[j+1])
      swap (a[j] , a[j+1]);
   }
 }
Void main()
 {
 int x[5] = { 8, 3, 2, 4, 1} ;
 char y[4] = {'g', 'y', 'd', 'a'} ;
  bubblesort (x,5);
  cout << "\n Sorted integers are\n";
for (int i=0; i<5; i++)
  cout << x[i] << " ";
bubblesort (y,4) ;
cout << "\n Sorted characters are\n";
  for (int i=0; i<4; i++)
   cout << y[i] << " ";
 }
```

o/p:

Sorted integers are

1 2 3 4 8

Sorted characters are

a d g y

(Eg) Bubble sort

	48	32	19	56	28	
Pass1 :	32	19	48	28	56	
Pass2 :	19	32	28	48	56	
Pass3 :	19	28	32	48	56	← Finalized
Pass4 :	19	28	32	48	56	

Generic function for searching algorithm

Linear search

```
#include <iostream.h>
template <class T>
void lsearch ( T *a , T item , int n)
 {
  int z=0;
  for(int i=0; i<n; i++)
   {
   if (a[i] == item)
    {
     z=1;
    cout << "Item found at position: "<< (i+1)<<"\n";
    }
  }
  if (z==0)
    cout << " Item not found in the list\n";
}
void main()
{
int x[6] = { 72, 42, 36, 86, 87, 29};
double y[4] = {6.4, 5.53, 44.4, 89.57};
int x1;
double y1;
cout << "Elements of Integer Array \n";
 for (int i=0; i<6; i++)
   cout <<" " << x[i] <<" ";
 cout << "Enter the item to be searched:\n";
 cin >> x1;
  lsearch (x, x1, 6);
 cout << "Elements of double Array \n";
 for(int i=0; i<4; i++)
    cout << y[i] <<" ";
cout << "Enter an item
    to be searched \n
cin>>y1;
lsearch (y,y1,4);
}
```

Overloading of Function Templates

* Function templates can also be overloaded with multiple declarations.
* It may be overloaded by either
  - other functions of the same name or
  - other template functions of the same name.
* Overloaded functions differ in terms of number of parameters or their types.

(Eg)

```
#include <iostream.h>
template <class T>
void print (T data)  // Single template argument
 {
  cout << data << endl;
 }
template <class T>
void print (T data, int n) // template and standard argument.
 {
  for (int i=0; i<n; i++)
    cout << data << endl;.
 }
void main()
 {
  print (2);
  print (2.5);
  print (300, 2);
  print ("Hello", 3);
 }
```

o/p:

```
2
2.5
300
300
Hello
Hello
Hello
```

* Here required fn is selected based on no. of arguments at fn call.

Class Templates

* Classes can also be declared to operate on different data types. Such classes are called class templates.
* Thus class templates are generic classes which support similar operation for different data types.

Syntax of class template declaration:

```
template <class T1, class T2, ... >
class classname
 {
  // data items of template type T1, T2..
   T1 data1;
  // functions of template arguments T1,T2...
   void func1 (T1 a, T2 b);
    :
 };
```

Syntax of class template instantiation (or) object creation

```
classname <char> obj1 ;
classname <int> obj2 ;
```

(datatype to be substituted for template datatype)

Syntax for declaring member function of class template outside its body.

```
template <class T>
class classname
  {
    :
   void func1 (T a);
  };
template <class T>
Void classname <T> :: func1 (T a)
  {
    // fn body using template parameters.
  }
```

Implementation of Generic stack using Class Templates

```
#include<iostream.h>
#define MAX 10
template <class T>
class stack
{
protected:
  T arr[MAX];
public:
   T item, r;
   int top;
   stack()
   {
    top = -1;
   }
void push(T a)
 {
  top++;
  if(top < MAX)
   {
   arr[top] = a;
   }
  else
   {
   cout << "STACK IS FULL\n";
   top--;
   }
 }
T pop()
{
 if (top == -1)
  {
   cout << "STACK IS EMPTY\n";
   return NULL;
  }
 else
  {
   T data = arr[top];
   top--;
   return data;
  }
 }
};
void main()
{
stack <char> a;
int opt = 0;
do
{
cout << "MAX STACK CAPACITY:"
        << ((MAX - a.top) - 1) << "\n";
cout << "1. PUSH ITEM\n";
cout << "2. POP ITEM\n";
cout << "3. EXIT\n";
cout << "OPTION:\n";
cin >> opt;
switch(opt)
{
case 1:
cout << "Enter item to be pushed\n";
 cin >> a.item;
 a.push(a.item);
 break;
 case 2:
  a.r = a.pop();
 cout << "Item popped from Stack:"
                 << a.r << endl;
 getch();
 break;
}
}while(opt <= 2);
}
```