Unit – 6
Generic Programming

Templates

- A significant benefit of object oriented programming is reusability of the code which eliminates redundant coding.
  - Functions
  - Classes
- The method of Generic Programming is implemented to increase the efficiency of the code.
- Generic Programming enables the programmer to write a general algorithm which will work with all data types.
- It eliminates the need to create different algorithms if the data type is an integer, string or a character.

The advantages of Generic Programming are

- Code Reusability
- Avoid Function Overloading
- Once written it can be used for multiple times and cases.

Templates

- Type-independent patterns that can work with multiple data types.
  - Generic programming
  - Code reusable
- Function Templates
  - These define logic behind the algorithms that work for multiple data types.
- Class Templates
  - These define generic class patterns into which specific data types can be plugged in to produce new classes.

Function and function templates

- C++ routines work on specific types.
- We often need to write different routines to perform the same operation on different data types.

```
int maximum(int a, int b, int c)
{
   if(a>b)
          { if (a>c) return a;
           else return c; }
   else {
           if (b > c) return b;
          else return c; }
}
```

Function and function templates

- C++ routines work on specific types.
- We often need to write different routines to perform the same operation on different data types.

```
float maximum(float a, float b, float c)
{
   if(a>b)
         { if (a>c) return a;
           else return c; }
   else {
          if (b > c) return b;
         else return c; }
}
```

Function and function templates

```
float maximum(float a, float b, float c)
 {
   if(a>b)
         { if (a>c) return a;
           else return c; }
   else {
          if (b > c) return b;
         else return c; }
 }
```

- The logic is exactly the same, but the data type is different.
- Function templates allow the logic to be written once and used for all data types – generic function.

Function Templates

```
//Generic function to find a maximum value
#include<iostream>
using namespace std;
template <typename T>
T maximum(T a, T b, T c)
 {
    if(a>b)
          { if (a>c) return a;
            else return c; }
    else {
           if (b > c) return b;
          else return c; }
 }
```

```
int main()
 {
 int a,b,c;
 cout<<"Input three integer  number : ";
 cin>>a>>b>>c;
 int d = maximum(a,b,c);
 cout<<"Maximum of three values is : "<<d;

 float a1,b1,c1;
 cout<<"Input three float number : ";
 cin>>a1>>b1>>c1;
 float d1 = maximum(a1,b1,c1);
 cout<<"Maximum of three values is : "<<d1;

 char a2,b2,c2;
 cout<<"Input three characters : ";
 cin>>a2>>b2>>c2;
 char d2 = maximum(a2,b2,c2);
 cout<<"Maximum of three values is : "<<d2;
 }
```

- ❖ Template function itself is incomplete because the compiler will need to know the actual type to generate code
- ❖ C++ compiler will then generate the real function based on the use of the function template.
- ❖ Each call to `maximum()` on a different data type forces the compiler to generate a different function using the template.

Function Templates

```
//Generic function to find a maximum value
#include<iostream>
using namespace std;
template <class T>
T maximum(T a, T b, T c)
 {
   if(a>b)
         { if (a>c) return a;
           else return c; }
   else {
          if (b > c) return b;
         else return c; }
 }
```

```
int main()
 {
 int a,b,c;
 cout<<"Input three integer  number : ";
 cin>>a>>b>>c;
 int d = maximum(a,b,c);
 cout<<"Maximum  of three  values  is : "<<d;

 float a1,b1,c1;
 cout<<"Input three float number : ";
 cin>>a1>>b1>>c1;
 float d1 = maximum(a1,b1,c1);
 cout<<"Maximum  of three  values  is : "<<d1;

 char a2,b2,c2;
 cout<<"Input three integer  number : ";
 cin>>a2>>b2>>c2;
 char d2 = maximum(a2,b2,c2);
 cout<<"Maximum  of three  values  is : "<<d2;
 }
```

- Template function itself is incomplete because the compiler will need to know the actual type to generate code
- C++ compiler will generate the real function based on the use of the function template.
- Each call to `maximum()` on a different data type forces the compiler to generate a different function using the template.

Example

```
template <typename T, typename T1, typename T2>
void print_type(T a,  T1 b,  T2 c)
 {
   cout<<" a is integer :" <<a;
   cout<<" b is float :" <<b;
   cout<<" c is character  :" <<c;
 }
int main()
{              int a; float b; char c;
cout<<"Input  integer number : ";  cin>>a;
cout<<"Input  float :";   cin>>b;
cout<<"Input  character :";   cin>>c;
print_type(a,b,c);       }
```

```
int main()
 {            int a; float b; char c;
 cout<<"Input integer number : ";  cin>>a;
 cout<<"Input float :";    cin>>b;
 cout<<"Input character :";    cin>>c;
 print_type(c,a,b);
}
```

Class template

- Templates are used to parameterize classes and functions with different types
  - Function templates allow writing generic functions that work on many types.
  - Function templates do not require explicit declaration of parameterized types when called
    - E.g., maximum(i, j, k);
  - Same idea applies to defining generic classes that work with many types -- extract the type to be a template to make a generic classes.
  - Classes require explicit declaration of parameterized types when instantiated
    - E.g., vector <int> v;
- Some compilers require that template definitions be included with declarations

Class template

To make a class into a template, prefix the class definition with the syntax:

template< class T >

- Here T is just a type parameter. Like a function parameter, it is a place holder.
- When the class is instantiated, T is replaced by a real type.

To access a member function, use the following syntax:

- className < T > object.
- object.SimpleList()

Using the class template:

- ClassName <real type> variable;
- SimpleList < int > list1;

Example (using constructor)

```
template <typename  T>
class Array  {
private:
   T* ptr;
   int size;

public:
   Array(T  arr[], int s)
   {
   ptr = new T[s];
   size = s;
   for (int i = 0; i < size;  i++)
      ptr[i] = arr[i];
   }
   void print()
   {
   for (int i = 0; i < size;  i++)
   cout << " " << *(ptr + i);
   cout << endl;
   }
};
```

```
int main()
{
   int arr[5] = { 1, 2, 3, 4, 5 };
   Array <int> a(arr, 5);
   a.print();

   float arr1[] = {2.3, 2.4, 3.4, 4.5, 5.6};
   Array <float> a1(arr1,5);
   a1.print();
   return 0;
}
```

Example (using member function)

```
template <class T>
class Array {
private:
   T* ptr;
   int size;

public:
   void getdata(T  arr[], int s)
   {
   ptr = new T[s];
   size = s;
   for (int i = 0; i < size; i++)
      ptr[i] = arr[i];
   }
   void print()
   {
   for (int i = 0; i < size; i++)
   cout << " " << *(ptr + i);
   cout << endl;
   }
};
```

```
int main()
{
   int arr[5] = { 1, 2, 3, 4, 5 };
   Array <int> a;
   a.getdata(arr, 5);
   a.print();

   float arr1[] = {2.3, 2.4, 3.4, 4.5, 5.6};
   Array <float> a1;
   a1.getdata(arr1,5);
   a1.print();
   return 0;
}
```

Problem statement for class template

Write a generic program to multiply vector with scalar quantity for integer data as well as double data (real data). Use class template for generic code.

Create class vector to define data in one dimensional array (this could be generic data type).

Write member function to initialize data of vector object as integer or double as per requirement.

Write another function to multiply vector object with scalar quantity and display the resultant vector.

Also write a function to add integer vector with double and display the result.

Problem statement for function template

Write a generic program to find out the standard deviation of marks (int) of the OOP course or total percentage (float) of the class. Use function template for generic code.

Write a function template that accepts marks (int) or percentage (float) of all the students and calculate standard deviation.

Standard deviation = sqrt (variance)

Variance = $\sum_{k=0}^{n-1}(x^k - mean)^2$ / n

Mean = $\sum_{k=0}^{n-1}(x^k)\ /n$

Also compare standard deviation of OOP marks with standard deviation of percentage, display the largest one.