Templates

Lecture 20

Templates

- Powerful software reuse feature of C++
  - Function templates – specify with a single code segment an entire range of related (overloaded) functions
  - Class templates – specify with a single code segment an entire range of related classes

- ***Generic programming***

Function overloading

- Perform similar or identical operations on different types of data

```
void add(int a,int b)
{ cout<<(a+b); }
void add(float a,float b)
{ cout<<(a+b); }
```

Note – operation performed in the two are identical, that is to add the two arguments and print the result

- These type of overloaded functions can be expressed more compactly and conveniently using function templates

Function templates

```
#include<iostream.h>
#include<conio.h>
template <class T>
void add(T a,T b)
{
   cout<<"\n Sum is : "<<(a+b);
}
void main()
{
 add(5,10);
 add(2.5,3.5);
 }
```

Compiler side

- Based on the argument types provided explicitly or inferred from calls to this function, the compiler generates separate object-code functions to handle each call separately
- Size of object file changes according to the number of types used to call the function

Overloading function templates

- A function template also can be overloaded by providing non-template functions with the same function name but different function arguments

```
template <class T>
void add(T a,T b)
{    cout<<"\n Sum is : "<<(a+b); }
void add(char a,char b)
{ cout<<“\n Concatenation :”<<a<<b; }
void main()
{ add(5,10); add(2.5,3.5); add(‘f’,’g’);  }
```

compiler

- Compiler uses overloading resolution to invoke the proper function
- It first finds all function templates and ordinary functions that best match the call
- If both template and ordinary function matches, then ordinary is called

A function with two generic types

```
template <class T1,class T2>
void display(T1 a,T2 b)
{    cout<<a<<b; }
void main()
{ display(10,'a')  }
```

Generic classes

- Type of data (member) can be generic
- Actual type can be specified at the time of making the object of that class
- Also know as parameterized types
- Are useful when a class uses logic that can be generalized, for example stack

Example

```
template <class T>
class mytempclass
{
  private:
  T x;
  public:
  void getdata() { cout<<"\n Enter data " ; cin>>x; }
  void display() { cout<<"\n x : "<<x; }
};
void main()
{ mytempclass<int> obj;
   obj.getdata();   obj.display();
  mytempclass<char> obj2; obj.getdata(); obj.putdata();
}
```

Class exercise

- Write a function template isEqual that compares its two arguments of the same type, which return true/false

- Also write main function in which this function is used for the built in types (int, float and char)