Templets

Templets

Templets can be used to create a family of classes or functions

- Here's a small function that you might write to find the maximum of two integers.

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

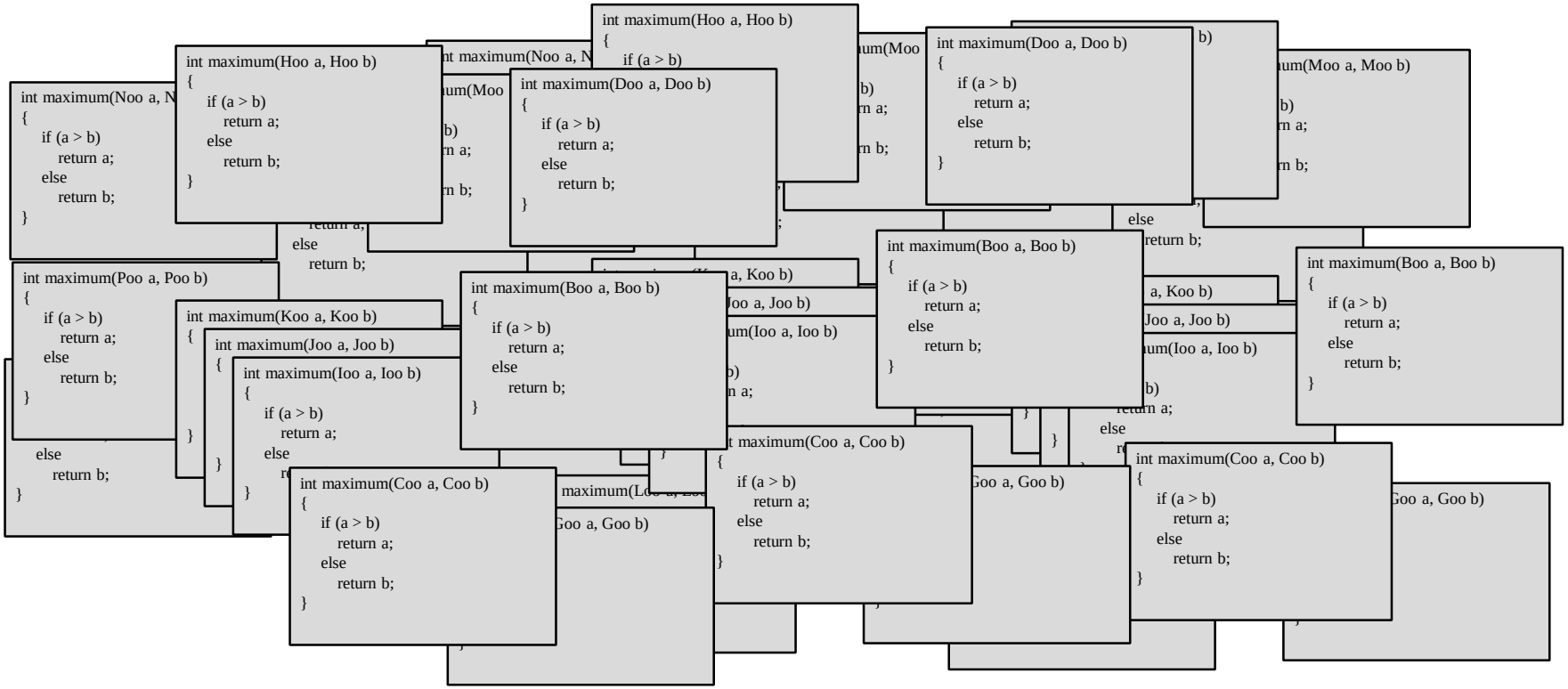
- Here's a small function that you might write to find the maximum of two double numbers.

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

- Here's a small function that you might write to find the maximum of two nitw.

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

- Suppose your program uses 100,000,000 different data types, and you need a maximum function for each...



•template<class T>

•template<typename T>

Returntype functionname(parameters)

T SUM(T a)

The templated function works using either the explicit or implicit template expression

square<int>(value) or square(value).

Parametrized types

A Template Function for Maximum

- This template function can be used with many data types.

```
template <class T>
T myMax(T x, T y)
{
  return (x > y)? x: y;
}

int main()
{
  cout << myMax<int>(3, 7) << endl;  // Call myMax for int
  cout << myMax<double>(3.0, 7.0) << endl; // call myMax for double
  cout << myMax<char>('g', 'e') << endl;   // call myMax for char

  return 0;
}
```

```
int myMax(int x, int y)
{
  return (x > y)? x: y;
}
```

```
char myMax(char x, char y)
{
  return (x > y)? x: y;
}
```

Using a Template Function

- Once a template function is defined, it may be used with any adequate data type in your program...

```
template <class Item>
Item maximum(Item a, Item b)
{
    if (a > b)
        return a;
    else
        return b;
}
```

```
cout << maximum(1,2);
cout << maximum(1.3, 0.9);
...
```

Class Templates

class implementation that is same for all classes, only the data types used are different.

class templates make it easy to reuse the same code for all data types.

```
template <class T>
 class className
 { ... .. ...
     public:
      T var;
      T someOperation(T arg);
      ... .. ...
};
```

Class template object

```
className<dataType>classObject;
```

For example:

```
className<int>classObject;
className<float>classObject;
className<string>classObject;
```

How the memory allocate to object?
Based on variable and data types. If it is integer for 2 variables it take 4 bytes and for float 8 bytes etc.

For template class we include datatype at the time of object creation.

Simple calculator using Class template

```
template <class T>
class Calculator
{
private:
        T num1, num2;
public:
        Calculator(T n1, T n2)
        {
                num1 = n1;
                num2 = n2;
        }
void displayResult()
{
         cout << "Numbers are: " << num1 << " and " << num2 << "." << endl;
        cout << "Addition is: " << add() << endl;
         cout << "Subtraction is: " << subtract() << endl;
        cout << "Product is: " << multiply() << endl;
        cout << "Division is: " << divide() << endl;
}
```

```
T add()
{
        return num1 + num2;
}
T subtract()
{
        return num1 - num2;
 }
T multiply()
 {
        return num1 * num2;
}
 T divide()
{
        return num1 / num2;
}
 };
```

```
int main()
{
Calculator<int> intCalc(2, 1);
Calculator<float> floatCalc(2.4, 1.2);
cout << "Int results:" << endl;
intCalc.displayResult();
 cout << endl << "Float results:" << endl;
floatCalc.displayResult();
return 0;
}
```

Int results:
Numbers are: 2 and 1.
Addition is: 3
Subtraction is: 1
Product is: 2
Division is: 2
Float results:
Numbers are: 2.4 and 1.2.
Addition is: 3.6
Subtraction is: 1.2
Product is: 2.88
Division is: 2

If member function definition is inside class then its act as inline function

If member function definition is outside class then use again template key word.

```
template <class T>
class test
{
  T   a, b;
 public:
  void getf()     //defination
{
   cin>>a>>b;
}
T   sum();      //declaration
};
```

```
template <class T> //def
// T  test :: sum()
T  test  <T> :: sum()

// it is template class mem fun
{
  return a+b;
}

void main()
{
```

```
// for multiple data types one is int other is char
template <class T, class U>
 T GetMin (T a, U b)
{
        return (a<b?a:b);
}
```

```
    int i,j;
     long l;
     i = GetMin<int,long> (j,l);
```

```
    i = GetMin (j,l);
```

for Different arrays

```
template <class T>
 T GetMin (T a[], int n)
```

Int n is for array size, array size is integer value only but array values may be int, float

for Different arrays

```
template <class T>
 T sum(T a[], int n)
{
   T    s=0;
   for(int i=0; i<n;  i++)
   {
     s=s+a[i];
    }
   return s;
}

Int main()
{
int  x[5]={10, 20, 30, 40, 50};
float  y[3]={1.1, 2.2, 3.3};
cout<<“int array elements sum=“<<sum(x, 5);
cout<<“float array elements sum=“<<sum(y, 3);
}
```

Overloading function Template

```
template <class T>

 T sum(T a, T b)
{
   return a+b;
}

template <class T>

 T sum(T a, T b, T c)
{
     return a+b+c;
}

Void main()
{
   cout<<“two integer sum=”<<sum(10, 20);
   cout<<“two float sum=“<<sum(1.1, 2.5);
   cout<<“three float sum=“<<sum(1.1, 2.5, 3.3);
    cout<<“three integer sum=”<<sum(10, 20, 30);
}
```

```
// template specialization

template <class T>
  T square(T x)
  {
       T result;
       result = x *x;
       return result;
  }

  template < >
  string square<string>(string ss)
  {
     return (ss+ss);
  };
```

Specify a default type parameter and default non-type parameter:

```
template <typename T=float, int count=3>
```

Minor 2 Topics

Constructors and Destructors

friend functions

Classes and Objects Passing Objects

Inheritance

Polymorphism

Storage classes

Modifiers and Qualifiers

Templets