Functions II

NEW YORK UNIVERSITY
ABU DHABI

- Recursive Function
- Function Overloading
- Function templates

Recursive Function

NEW YORK UNIVERSITY
ABU DHABI

- Recursive Function
- Function Overloading
- Function templates

NEW YORK UNIVERSITY

ABU DHABI

A recursive function in C++ is a function that calls itself.

```
void foo(int nValue)
{
    //Statements
    foo(nValue-1);
}
```

Function calls itself

```
void foo(int nValue)
{
    using namespace std;
    cout << nValue << endl;
    foo(nValue-1);
}

int main(void)
{
    foo(12);
    return 0;
}
```

Termination Condition

```
void foo(int nValue)
{
    using namespace std;
    cout << nValue << endl;

    // Termination Condition
    if(nValue > 0)
        foo(nValue-1);
}

int main(void)
{
    foo(12);
    return 0;
}
```

NEW YORK UNIVERSITY

ABU DHABI

```
// return the sum of 1 to nValue

int Sum(int nValue)
{
    if (nValue <=1)
        return nValue;
    else
        return Sum(nValue - 1) + nValue;
}

int main(void)
{
    Sum(12);
    return 0;
}
```

```
Sum(3) called, 3 <= 1 is false, so we return Sum(2) + 3.
Sum(2) called, 2 <= 1 is false, so we return Sum(1) + 2.
Sum(1) called, 1 <= 1 is true, so we return 1.
This is the termination condition.

Now we unwind return process:
Sum(1) returns 1.
Sum(2) returns Sum(1) + 2, which is 1 + 2 = 3.
Sum(3) returns Sum(2) + 3, which is 3 + 3 = 6.

Consequently, Sum(3) returns 6.
```

Function Overloading

NEW YORK UNIVERSITY
ABU DHABI

- Recursive Function
- Function Overloading
- Function templates

NEW YORK UNIVERSITY

ABU DHABI

Function overloading:

C++ that allows us to define multiple functions with the same name with different parameters.

```
int MultiplyI(int nX, int nY)
{
    return nX * nY;
}
```

A function of multiplying two integers

What if we want to multiplying two floating numbers?

```
int MultiplyI(int nX, int nY)
{
    return nX * nY;
}

double MultiplyD(double dX, double dY)
{
    return dX * dY;
}
```

One Solution is to define two different functions

NEW YORK UNIVERSITY

ABU DHABI

```
double Multiply(double dX, double dY, double dZ)
{
    return dX * dY * dZ;
}
```

Declare another Multiply() function that takes double parameters

Function Templates

NEW YORK UNIVERSITY
ABU DHABI

- Recursive Function
- Function Overloading
- Function templates

NEW YORK UNIVERSITY ABU DHABI

Function templates are functions that serve as a pattern for creating other similar functions.

```
int add(int nX, int nY)
{
    return nX + nY;
}
```

There are 3 places where specific types are used: parameters nX, nY, and the return value all specify that they must be integers.

To create a function template, we will replace these specific types with placeholder types

```
Type add(Type tX, Type tY)
{
    return tX + tY;
}
```

However, it won't compile because the compiler doesn't know what "Type" means!

```
Type add(Type tX, Type tY)
{
    return tX + tY;
}
```

In order to tell the compiler that Type is meant to be a placeholder type, we need to formally tell the compiler that Type is a template type parameter. This is done using what is called a template parameter declaration:

```
// this is the template parameter declaration
template <typename Type>
Type add(Type tX, Type tY)
{
    return tX + tY;
}
```

If the function uses multiple template type parameter, they can be separated by commas:

```
template <typename T1, typename T2>
// template function here
```

For example:

```
// this is the template parameter declaration
template <typename T1, typename T2>
T1 add(T1 tX, T2 tY)
{
    return tX + tY;
}
```

```
int nValue = add(3, 7);
double dValue = add(6.34, 18.523);
char chValue = add('a', '6');
```