Introduction to C++ Programming

Its Applications in Finance

Thanh Hoang

Claremont Graduate University

October 3, 2012

Today Agenda

1. Function Fundamentals
   - □ General Form
   - □ Create a Function
   - □ Using Arguments and *return* Statement
2. Function Prototypes
3. Scope
   - □ Local Scope
   - □ Global Scope
4. Passing to Functions
   - □ Pointers in Functions
   - □ Arrays in Functions
   - □ Strings in Functions
5. Recursion
6. Inline Functions
7. Overloaded Functions
8. Summary

Function

General Form

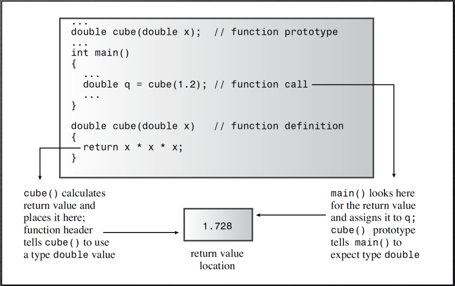
type functionName(*parameterList*) {

// function body

}

Definition

1. Function contains the lines of code that performs a specific assignment.
2. Prototype defines function name, argument types, and return value.
3. Call causes the function to be executed.



```
...
double cube(double x);  // function prototype
...
int main()
{
  ...
  double q = cube(1.2); // function call
  ...
}

double cube(double x)   // function definition
{
  return x * x * x;
}
```

cube() calculates return value and places it here; function header tells cube() to use a type double value

1.728

return value location

main() looks here for the return value and assigns it to q; cube() prototype tells main() to expect type double

Define, Prototype and Call a Function

```
#include <iostream>
using namespace std;

// Function Prototype
void myfunc();

int main()
{
    cout << "main() will call the myfunc() function: ";
    cout << endl;

    // Calls a function
    myfunc();

    return 0;
}

// Function Definition
void myfunc() {
    cout << "I m a simple function." << endl;
}
```

```
main() will call the myfunc() function:
I m a simple function.
```

Demonstrate a Function

```cpp
#include <iostream>
using namespace std;

void volume(int length, int width, int height);  // Function prototype

int main()
{
    volume(10, 9, 8); // Calls functions
    volume(9, 8, 7);
    volume(8, 7, 6);

    return 0;
}

// Function definition - Computes the volume of a box
void volume(int length, int width, int height) {
    cout << "The volume of a box is: " << length * width * height << endl;
}
```

```
The volume of a box is: 720
The volume of a box is: 504
The volume of a box is: 336
```

return Statement

General Form

The *return* has two different forms:

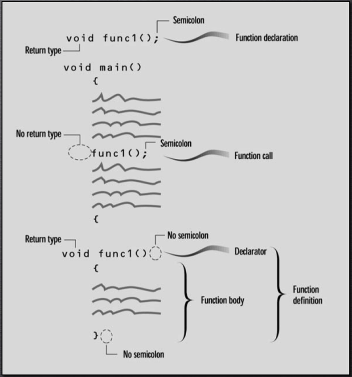
1. A *return* statement does not return a value (*void*).

   ```
   void functionName(parameterList)  {
       statement(s)
       return; // optional
   }
   ```

2. A *return* statement does return a value (*return-type*).

   ```
   type functionName(parameterList)  {
       statement(s)
       return value;
   }
   ```

Note: The return value cannot be an array.



```
void func1();

void main()
{

func1();

{

void func1()
{

}
```

return Statement (*void*)

```cpp
#include <iostream>
using namespace std;

void power(int base, int hat); // power() prototype

int main()
{
    power(10, 2); // Call functions
    power(10, -2);

    return 0;
}

// power() function raises an integer to a positive integer
void power(int base, int hat) {
    int temp=1;

    if(hat < 0)
        return; // power() function cannot do a negative exponents
    else
        for( ; hat; hat--) temp *= base;
    cout << "The answer is: " << temp << endl;
}
```

return Statement (*value*)

Definition

A return value is a way to get information out of a function.

General Form

return value;

```
#include <iostream>
using namespace std;

int volume(int length, int width, int height); // volume() prototype

int main()
{
    int value = volume(8, 9, 10); // Calls a function
    cout << "The volume is: " << value << endl;

    return 0;
}

// volume() function
int volume(int length, int width, int height) {
    return length * width * height;
}
```

Functions in Expressions

```cpp
#include <iostream>
using namespace std;

double volume(double l, double w, double h); // volume() prototype

int main()
{
    double total;

    total = volume(10.9, 9.8, 8.7) + volume(7.6, 6.5, 5.4) + volume(4.3, 3.2, 2.1);

    cout << "The sum of the volumes is: " << total << endl;
    cout << "The average of volumes is: " << total / 3 << endl;

    return 0;
}

double volume(double l, double w, double h) { // volume() definition
    return l * w * h;
}
```

```
The sum of the volumes is: 1224.99
The average of volumes is: 408.33
```

Function Prototype

Three Aspects of a Function

1. Function's return type
2. Type of its parameters
3. Number of its parameters

General Form

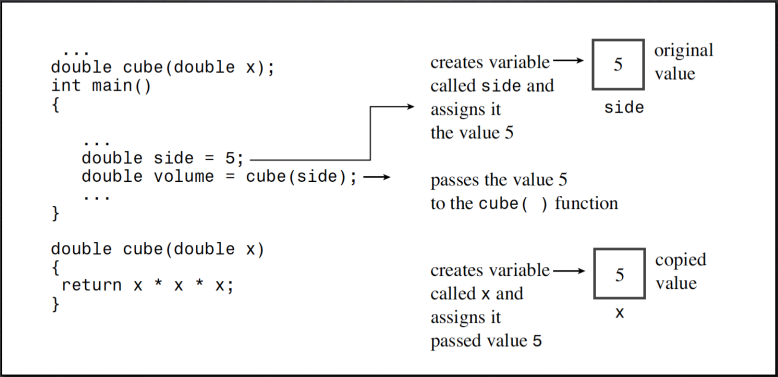
type myfunc(type *para1*, type *para2*, ..., type *paraN*);

Prototype a Function

```cpp
#include <iostream>
using namespace std;

// cheers() prototype: does not return value
void cheers(int n);

// cube() prototype: returns a double
double cube(double x);

int main()
{
    double num, volume;

    cheers(5); // Calls a function
    cout << "Give me a number: ";
    cin >> num;

    volume = cube(num); // Calls another function
    cout << "A " << num << "-foot cube has a volume of ";
    cout << volume << " cubic feet.\n";
    cheers(cube(2));

    return 0;
}
```

```cpp
// cheers() definition
void cheers(int n) {
    for(int i=0; i<n; i++)
        cout << "Cheers! ";
    cout << endl;
}

// cube() definition
double cube(double x) {
    return x * x * x;
}
```

Passing by Value



```
 ...
double cube(double x);
int main()
{

   ...
   double side = 5;
   double volume = cube(side);
   ...
}

double cube(double x)
{
 return x * x * x;
}
```

creates variable called side and assigns it the value 5

5

side

original value

passes the value 5 to the cube( ) function

creates variable called x and assigns it passed value 5

5

x

copied value

Omitting Function Prototype

```
#include <iostream>
using namespace std;

// Using a function s definition as its prototype
// isEven() determines whether a number is even
bool isEven(int a) {
    // Checks whether it is even or not
    if (!(a % 2))
        return true;
    else
        return false;
}
```

```
int main()
{
    int num;
    char key;

    do {
        cout << "Enter a number: ";
        cin >> num;

        if(isEven(num))
            cout << num << " is an even number." << endl;
        else
            cout << num << " is an odd number." << endl;

        cout << "Do another (y/n): ";
        cin >> key;

    } while(key != n );

    return 0;
}
```

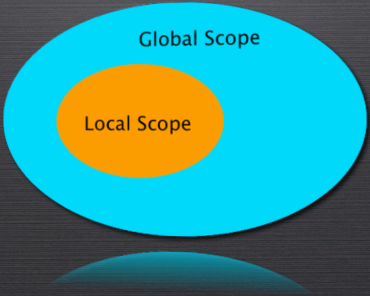
```
Enter a number: 4
4 is an even number.
Do another (y/n): y
Enter a number: 5
5 is an odd number.
Do another (y/n): n
```

C++ Scope

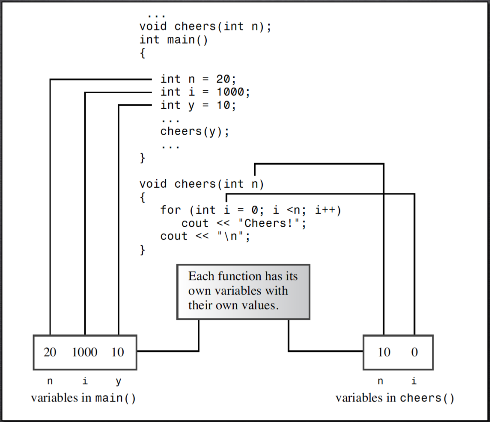
Definition

In general, the scope rules of a language govern the visibility and lifetime of an object.

1. Local Scope: Variables are visible only within a block.
2. Global Scope: Variables are visible throughout an entire program.



Local Scope



```
...
void cheers(int n);
int main()
{
   int n = 20;
   int i = 1000;
   int y = 10;
   ...
   cheers(y);
   ...
}

void cheers(int n)
{
   for (int i = 0; i <n; i++)
      cout << "Cheers!";
   cout << "\n";
}
```

Each function has its own variables with their own values.

20	1000	10
n	i	y

variables in main()

10	0
n	i

variables in cheers()

Local Scope (*cont.*)

```
#include <iostream>
using namespace std;

void myfunc(); // myfunc() prototype

int main()
{
    int var = 100; // a local variable in main() function

    cout << "var in main() function: " << var << endl;
    myfunc();
    cout << "var in main() function: " << var << endl;

    return 0;
}

void myfunc() {
    int var = 99; // a local variable in myfunc() function
    cout << "var in myfunc() function: " << var << endl;
}
```

```
var in main() function: 100
var in myfunc() function: 99
var in main() function: 100
```

Local Scope (*cont.*)

```
.........
void myfunc() {
    int myvar1;     // local variables in myfunc()
    float myvar2;

    myvar1 = 8;     // OK
    myvar2 = 9.0;   // OK
    yourvar = 10;   // Error: not visible in myfunc()
}
.........

void yourfunc() {
    int yourvar;    // local variable in yourfunc()

    myvar1 = 18;    // Error: not visible in yourfunc()
    myvar2 = 19;    // Error: not visible in yourfunc()
    yourvar = 20;   // OK
}
.........
```

Name Hiding

```cpp
#include <iostream>
using namespace std;

int main()
{
    int a=9, b=10;

    if (b>0) {
        int a; // local variable in the if statement
        a = b * 2;
        cout << "Inner a: " << a << endl;
    }

    cout << "Outer a: " << a << endl;

    return 0;
}
```

```
Inner a: 20
Outer a: 9
```

Global Scope

```
#include <iostream>
using namespace std;

// myfunc() prototype
void myfunc();

// otherfunc() prototype
void otherfunc();

int a; // a global variable

int main()
{
    int i; // a local variable in main()

    for(i=0; i<5; i++) {
        a = i * 10;
        myfunc();
    }

    return 0;
}
```

```
void myfunc() { // myfunc() definition
    cout << "a: " << a;
    otherfunc();
}

void otherfunc() { // otherfunc() definition
    for(a=0; a<5; a++) cout << "*";
    cout << endl;
}
```

```
a: 0*****
a: 10*****
a: 20*****
a: 30*****
a: 40*****
```

Passing a Pointer to a Function

```cpp
#include <iostream>
using namespace std;

void myfunc(int *i); // myfunc() prototype

int main()
{
    int a, *ptr;
    ptr = &a; // ptr points to a

    myfunc(ptr); // passing a pointer ptr to myfunc()
    cout << "a: " << a << endl; // a is now 10

    return 0;
}

// myfunc() receives an integer pointer ptr
void myfunc(int *i) {
    *i = 10;
}
```

```
a: 10
```

Passing an Array to a Function

```
#include <iostream>
using namespace std;

void myfunc(int num[10]); // myfunc() prototype

int main()
{
    int a[10], t;
    for(t=0; t<10; t++) a[t] = t;

    myfunc(a); // passing an array to a function
    cout << endl;

    return 0;
}

// myfunc() prints out the values in an array
void myfunc(int num[10]) {
    for(int t=0; t<10; t++) cout << num[t] << " ";
}
```

```
0 1 2 3 4 5 6 7 8 9
```

Passing a String to a Function

```cpp
#include <iostream>
#include <cstring>
#include <cctype>
using namespace std;

void myfunc(char *ptr); // myfunc() prototype

int main()
{
    char str[80];
    strcpy(str, "I Love Programming");
    myfunc(str);
    cout << str << endl;

    return 0;
}

void myfunc(char *ptr) { // myfunc() inverts the case of letters in a string
    while(*ptr) {
        if (isupper(*ptr)) *ptr = tolower(*ptr);
        else *ptr = toupper(*ptr);

        ptr++; // moves on to a next letter via a pointer
    }
}
```

```
i lOVE pROGRAMMING
```

Recursion

Definition

Recursion is the process of a function calling itself. The process will run forever unless we include something to terminate the chain of calls in our code. Usually we use an *if* statement.

General Form

```
type recursionName((parameterList)  {
     statements1;
     if (test)
          recursionName(parameterList);
     statements2;
}
```

```cpp
#include <iostream>
using namespace std;

void countdown(int n); // countdown() prototype

int main()
{
    countdown(4); // Calls the recursive function
    return 0;
}

void countdown(int n) { // countdown () definition
    cout << "Counting down ... " << n << endl;
    if (n > 0)
        countdown(n-1); // function calls itself
    else
        return;
}
```

```
Counting down ... 4
Counting down ... 3
Counting down ... 2
Counting down ... 1
Counting down ... 0
```

Factorial Function by Recursion

fact()

Write a C++ program that asks the user to enter an integer number, and then the program displays its factorial.

Assignment

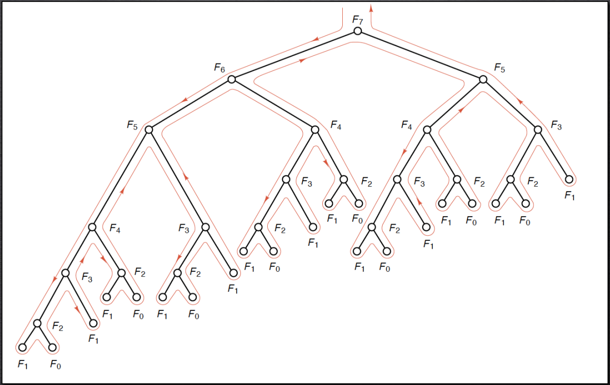
Your task is to create a function *fact*(), which calculates the factorial of an integer by using the recursion method.

```
int fact(int a) { // fact{} definition
    int result;

    if (a==1)
        result = 1;
    else
        result = fact(a-1) * a; // recursion

    return result;
}
```

```
Enter a number: 1
1! is 1
Do another (y/n): y
Enter a number: 5
5! is 120
Do another (y/n): n
```

Fibonacci Function by Recursion

Figure : Fibonacci Function by Recursion



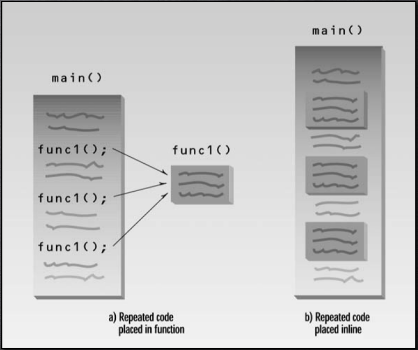
Inline Function

Purpose

Inline function is a C++ enhancement designed to speed up programs, but may require more memory than normal functions unless they are very small.

General Form

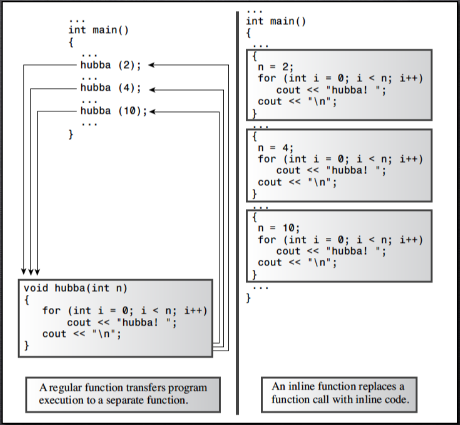
```
inline type functionName(parameterList)  {
      // function body
}
```



a) Repeated code placed in function

b) Repeated code placed inline

Example of Inline Function



```
...
int main()
{
    ...
    hubba (2);
    ...
    hubba (4);
    ...
    hubba (10);
    ...
}
```

```
void hubba(int n)
{
    for (int i = 0; i < n; i++)
        cout << "hubba! ";
    cout << "\n";
}
```

A regular function transfers program execution to a separate function.

```
...
int main()
{
 ...
 {
   n = 2;
   for (int i = 0; i < n; i++)
       cout << "hubba! ";
   cout << "\n";
 }
 ...
 {
   n = 4;
   for (int i = 0; i < n; i++)
       cout << "hubba! ";
   cout << "\n";
 }
 ...
 {
   n = 10;
   for (int i = 0; i < n; i++)
       cout << "hubba! ";
   cout << "\n";
 }
 ...
}
```

An inline function replaces a function call with inline code.

Inline Function

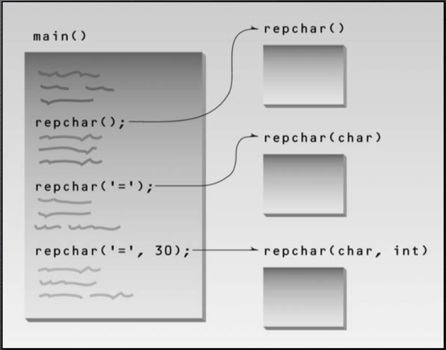
```
#include <iostream>
using namespace std;

inline double square(double); // Inline function prototype

int main()
{
    double a = square(5.0);
    double b = square(4.5 + 7.5); // can pass expressions
    double c = 13.0;

    cout << "a = " << a << ", b = " << b << endl;
    cout << "c = " << c << ", c squared = " << square(c++) << endl;
    cout << "Now c = " << c << endl;

    return 0;
}

inline double square(double x) { // Inline function definition
    return x * x;
}
```

```
a = 25, b = 144
c = 13, c squared = 169
Now c = 14
```

Overloaded Functions

Definition

An overloaded function is a group of functions with the same name, and each function is called depends on the type and number of arguments supplied in the call.



main()

repchar();

repchar('=');

repchar('=', 30);

repchar()

repchar(char)

repchar(char, int)

Example of Overloaded Functions

```cpp
#include <iostream>
using namespace std;

// Function Prototypes
void repchar();
void repchar(char ch);
void repchar(char ch, int n);

int main()
{
    repchar();
    repchar('=');
    repchar('+', 30);

    return 0;
}
```

```cpp
// Displays 45 asterisks
void repchar() {
    for (int j=0; j<45; j++) cout << "*";
    cout << endl;
}

// Displays 45 copies of specified character
void repchar(char ch) {
    for (int j=0; j<45; j++) cout << ch;
    cout << endl;
}

// Displays specified number of copies of specified character
void repchar(char ch, int n) {
    for (int j=0; j<n; j++) cout << ch;
    cout << endl;
}
```

```
*********************************************
=============================================
++++++++++++++++++++++++++++++
```

Summary

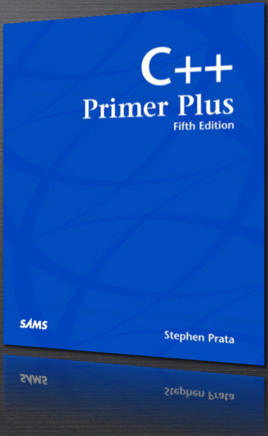
1. Function fundamentals
   - General form
   - Create a function
   - Using arguments and *return* Statement
2. Function prototype
3. Scope
4. Passing to functions
   - Pointers in functions
   - Arrays in functions
   - Strings in functions
5. Recursion
6. Inline Functions
7. Overloaded Functions

Reading

Stephen Prata.

C++ Primer Plus, 5th Edition,
Chapter 7

SAMS Publishing, 2004.



Overloaded Functions

Write a C++ program to simulate a uniform random number. In this program, you need to create two overloaded functions:

1. *runiform*(): generates a random uniform number in [0,1]
2. *runiform*(int a, int b): generates a random uniform number in [a,b], where:

   a: the beginning point of the interval

   b: the ending point of the interval

Area of a Circle

Write a C++ program to compute the area of a circle. In this program, you may need to create three functions:

1. *runiform*() gives us a random uniform number in the range [0,1].
2. *simPi*() gives us the simulated π; where n is the number of iterations for the simulation.
3. *circleArea*() computes the area of a circle.

Hint: The 2nd function needs to call the 1st function while the 3rd function needs to call the 2nd function to get the desired result.

Pricing American Option

Write a C++ program to price an American option by using the methodology of binomial approximation. You need to create two functions:

1. *computeAC*() calculates an American Call option.
2. *computeAP*() calculates an American Put option.

For simplicity, we consider a 2-year American put option with a strike price of \$52 on a stock whose current price is \$50. We suppose that there are two time steps of one year, and in each time step, the stock price either moves up by 20% or move down by 20%. The risk-free interest rate in this case is 5%.

Given that the value of the risk-neutral probability:

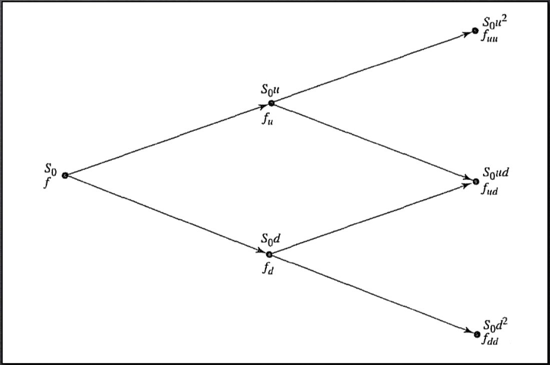
$$\begin{aligned} p &= \frac{e^{r\Delta t} - d}{u - d} \\ f &= e^{-r\Delta t}\left(pf_u + (1 - p)f_d\right) \end{aligned}$$

Pricing American Option (*cont.*)

Table : Example Data

Item	Value	Data Type
S_0	50.0	Float
K	52.0	Float
T	2	Integer
Δt	1	Integer
u	1.2	Float
d	0.8	Float
r	0.05	Float

Figure : Stock and Option Prices in a General Two-Step Binomial Tree


S_0
f
S_0u
f_u
S_0d
f_d
S_0u^2
f_{uu}
S_0ud
f_{ud}
S_0d^2
f_{dd}