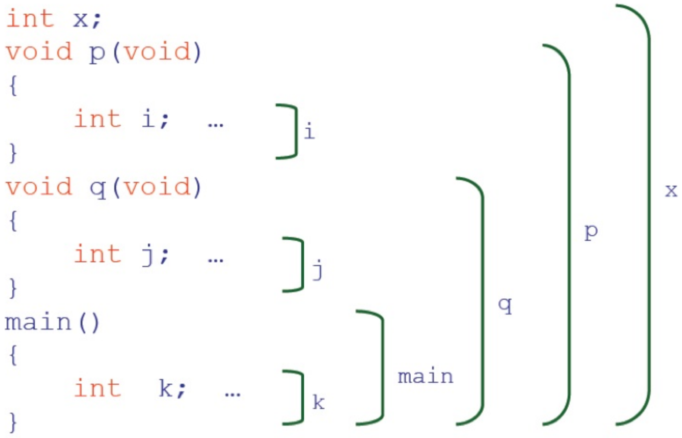
Functions

Yaping Jing

CS270 – Computer Science II

Variable Scope



```
int x;
void p(void)
{
     int i;  ...
}
void q(void)
{
     int j;  ...
}
main()
{
     int  k;  ...
}
```

Question

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

int main(){
  int len = 3;
  {
    cout << len << endl;
  }
}
```

Is there any error in the code?

Question

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

int main(){
  {
    int len = 3;
  }
  cout << len << endl;
}
```

Is there any error in the code?

Function Definition

```
return-type    function-name    ( 0 or more parameters )
{
    function-body
}
```

Return Type

int,   double,   char,   ...

Return Type

int,   double,   char,   ...

Return Type

int, double, char, ...

void

Function That Return Values

```
int celsius_to_fahrenheit(int celsius){
   int fahrenheit = celsius * 1.8 + 32;

   return fahrenheit;
}
```

Void Function

```
void print_temperature(int celsius){
   cout << celsius << endl;
}
```

Parameter List

Each parameter entry consists of **type** and **variable_name**; parameter entries are separated by a comma. e.g.

```
int computeArea (int x, double y);

void print_something( );
```

Function Body

```
int celsius_to_fahrenheit(int celsius){

    int fahrenheit = celsius * 1.8 + 32;

    return fahrenheit;
}
```

Function Declaration

signature, prototype, interface, ...

Function Prototype

return-type function-name (0 or more parameters);

```
int celsius_to_fahrenheit(int celsius);
```

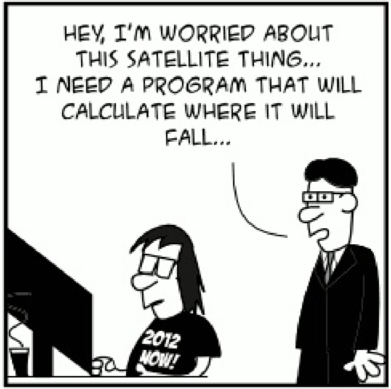
Function Declaration

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

int celsius_to_fahrenheit(int c);

int main(){
  cout << "Enter␣a␣temperature␣in␣celsius␣" << endl;
  int celsius;
  cin >> celsius;
  int fahrenheit = celsius_to_fahrenheit(celsius);
  cout << fahrenheit << endl;
  return 0;
}

int celsius_to_fahrenheit(int celsius){
   int fahrenheit = celsius * 1.8 + 32;
   return fahrenheit;
}
```

Function Declaration

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

int celsius_to_fahrenheit(int celsius){
   int fahrenheit = celsius * 1.8 + 32;
   return fahrenheit;
}

int main(){
  cout << "Enter␣a␣temperature␣in␣celsius␣" << endl;
  int celsius;
  cin >> celsius;
  int fahrenheit = celsius_to_fahrenheit(celsius);
  cout << fahrenheit << endl;
  return 0;
}
```

Precondition and Postcondition


HEY, I'M WORRIED ABOUT THIS SATELLITE THING... I NEED A PROGRAM THAT WILL CALCULATE WHERE IT WILL FALL...
2012 NOW!

Here are the requirements for the function...

What are Preconditions and Postconditions?

- One way to specify such requirements is with a pair of statements about the function.
- The **precondition** indicates what must be true before the function is called.
- The **postcondition** indicates what work the function has accomplished.

Specification Example

```
// Precondition: celsius >= -100.
// Postcondition: return temperature degree in fahrenheit.

int celsius_to_fahrenheit(int c);
```

Who Are Responsible for Pre/Post Conditions?

- Precondition is ensured by the programmer who calls the function.
- Postcondition is ensured by the programmer who write the function.

What if A Precondition Is Violated?

$assert(celsius >= -100);$ // #include<assert.h> header file

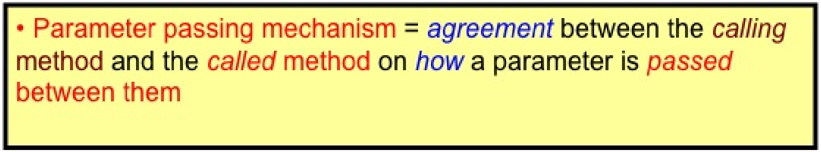
Assert Example

```
#include <iostream>
#include <assert.h>
using namespace std;

int celsius_to_fahrenheit(int celsius){

   assert(celsius >= -100);

   int fahrenheit = celsius * 1.8 + 32;
   return fahrenheit;
}
```

Exercise

Requirements: write a complete function (including function signature with pre/post condition and function definition) that takes two integers and computes their division. You're ensured that the denominator is either greater than 0 or less than 0. Also, give an example of client code how to use the function you defined.

Parameter Passing



• Parameter passing mechanism = *agreement* between the *calling* method and the *called* method on *how* a parameter is *passed* between them

- Pass by Value
- Pass by Reference

Pass by Value

```
#include <iostream>
#include <assert.h>
using namespace std;

void swap(int x, int y){
   int temp = x;
   x = y;
   y = temp;
}

int main(){
  int a = 10;
  int b = 20;
  swap(a, b);
  cout << a << "␣" << b << endl;
}
```

Pass by Reference

The agreement used in the pass by reference mechanism:

For the calling method:

- creates the parameter variables for the called method
- copies the reference(=address) of the actual argument into the formal parameter

For the called method:

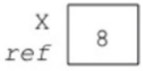
- uses the reference(=address) to locate the actual argument
- then it obtains the contents from the actual argument

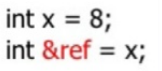
Variables

Variables	**Contents**	**Address**
x	5	FFF0
y	20	FFF1

Reference Variable

An **alias** for another variable.



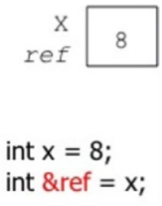


```
int x = 8;
int &ref = x;
```

```
cout << x << endl;
cout << ref << endl;
```

Reference Variable

An **alias** for another variable.

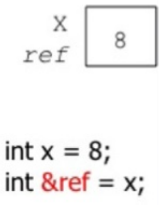


```
int x = 8;
int &ref = x;
```

```
x = 100;
cout << x << endl;
cout << ref << endl;
```

Reference Variable

An **alias** for another variable.



```
int x = 8;
int &ref = x;
```

```
x = 100;
ref = 200;
cout << x << endl;
cout << ref << endl;
```

Reference Variables in Functions

Defined with an ampersand (&) in both function prototype and function header.

```
void swap(int&, int& );
```

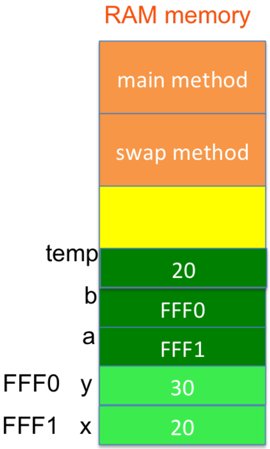
```
void swap(int& x, int& y)
```

Use Reference Variable as Parameter

```
void swap(int&, int& );


void swap(int& a, int& b) {
   int temp = a;
   a = b;
   b = temp;
}
```

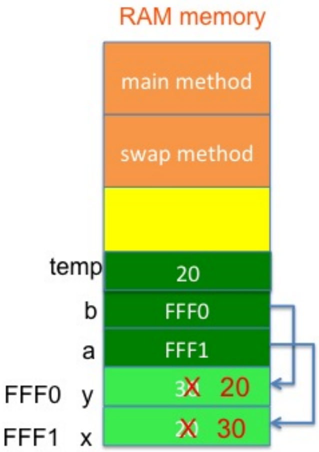
How Pass by Reference Work

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

void swap(int&, int& );

int main(){
  int x = 20;
  int y = 30;

  swap(x, y);
  cout << "x=" << x;
  cout << "␣y=" << y << endl;
  return 0;
}
void swap(int& a, int& b) {
      int temp = a;
      a = b;
      b = temp;
}
```



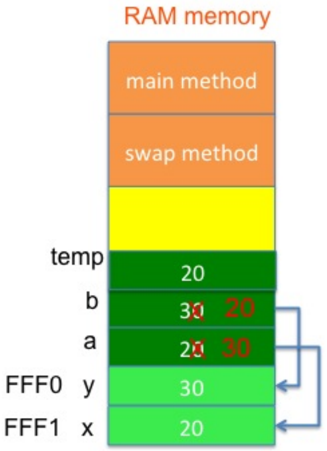
How Pass by Reference Work

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

void swap(int&, int& );

int main(){
  int x = 20;
  int y = 30;

  swap(x, y);
  cout << "x=" << x;
  cout << "␣y=" << y << endl;
  return 0;
}
void swap(int& a, int& b) {
       int temp = a;
       a = b;
       b = temp;
}
```



Pass by Value

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

void swap(int, int );

int main(){
  int x = 20;
  int y = 30;

  swap(x, y);
  cout << "x=␣" << x << ",␣y=␣" << y
  return 0;
}

void swap(int a, int b) {
        int temp = a;
        a = b;
        b = temp;
}
```



Only Variables May Be Passed By Reference

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

void swap(int&, int& );

int main(){

  swap(20, 30);  // error
  return 0;
}

void swap(int& x, int& y) {
       int temp = x;
       x = y;
       y = temp;
}
```

Only Variables May Be Passed By Reference

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

void swap(int&, int& );

int main(){
  int x = 10;
  int y = 30;
  swap(x+10, y);  // error
  return 0;
}

void swap(int& x, int& y) {
       int temp = x;
       x = y;
       y = temp;
}
```

A Mixture of Pass by Value and Pass by Reference

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

void f(int, int& );

int main(){
  int cat = 1;
  int dog = 5;
  f(cat, dog)
  cout << cat <<"␣" << dog << endl;
  return 0;
}

void f(int value, int& ref) {
       value++;
       ref++;
       cout<<value << "␣" << ref << endl;
}
```

Function Overloading

Two or more functions that have the same name, but different parameter lists.

```
int square (int num){
   return num*num;
}

double square (double num){
   return num*num;
}
```

An Example Using Overloading Functions

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

int square (int);
double square(double);

int main(){
  int myInt;
  double myFloat;

  cout << "Enter␣an␣integer␣and␣a␣floating-point␣value:␣";
  cin >> myInt >> myFloat;

  cout << "Here␣are␣their␣squares:␣";
  cout << square(myInt) << "␣and␣" << square(myFloat);

  return 0;
}
```

Function Signatures

The **function signature** is the name of the function and the data types of the function's parameters in the proper order.

```
square(int)

square(double)
```

Overloading Function – Quiz1

Do the following two functions have the same signature? [**Yes** or **No**]
Can we call them overloaded functions? [**Yes** or **No**]

```
int square (int num){
   return num*num;
}

double square (int num){
   return num*num;
}
```

Overloading Function – Quiz2

Do the following three functions have the same signature? [**Yes** or **No**]
Can we call them overloaded functions? [**Yes** or **No**]

```
int sum (int num, int num2)

int sum (int num, int num2, int num3)

int sum (int num, int num2, int num3, int num4)
```