

# Problem Solving With C++

**Mohammad A. Rajabi**

Dept. of Geomatics Eng.

University of Tehran

Cell: 0912 132 5823

Email: [marajabi@ut.ac.ir](mailto:marajabi@ut.ac.ir)

<http://www.marajabi.com>

# Calling Functions

- **There are three ways in C++ to pass arguments to a function:**
  - **call by value**
  - **call by reference with reference arguments**
  - **call by reference with pointer arguments**
- **When an argument is passed by value, a copy of the argument's value is made and passed to the called function**
- **Changes to the copy do not affect the original variable's value in the caller function.**

# Calling Functions

- **With call by reference, the caller gives the called function the ability to directly access the caller's data and to modify that data if the called function so chooses.**
- **To indicate that a function parameter is passed by reference, simply follow the parameter's type in the function prototype by an ampersand &.**
- **e.g. void myFunction( int &x)**

## Calling Functions

```
int squareByValue(int);
```

```
void squareByReference(int &);
```

```
main() {
```

```
    int x = 2, z = 4;
```

```
    cout << "x = " << x << " before squareByValue" << endl
```

```
        << "Value returned by squareByValue: "
```

```
        << squareByValue(x) << endl
```

```
        << "x = " << x << " after squareByValue" << endl << endl;
```

```
    cout << "z = " << z << " before squareByReference" << endl;
```

```
    squareByReference(z);
```

```
    cout << "z = " << z << " after squareByReference" << endl;
```

```
    return 0;
```

```
}
```

# Calling Functions

```
int squareByValue(int a)  
{  
    return a *= a; // caller's argument not modified  
}
```

```
void squareByReference(int &cRef)  
{  
    cRef *= cRef; // caller's argument modified  
}
```

## Call by Reference (using pointers)

```
void someFunction(int *x, int *y)
{
    int temp;

    temp = *x;
    *x = *y;
    *y = temp;
} // what does it do?
```

## Call by Reference (using pointers)

```
void swap(int *x, int *y);
```

```
int main() {
```

```
    int i, j;
```

```
    i = 10;
```

```
    j = 20;
```

```
    cout << "initial values of i and j: ";
```

```
    cout << i << ' ' << j << '\n';
```

```
    swap(&j, &i); // call swap() with addresses of i and j
```

```
    cout << "swapped values of i and j: ";
```

```
    cout << i << ' ' << j << '\n';
```

```
    return 0;
```

## Call by Reference (using pointers)

**// Exchange arguments.**

**void swap(int \*x, int \*y)**

**{**

**int temp;**

**temp = \*x; // save the value at address x**

**\*x = \*y; // put y into x**

**\*y = temp; // put x into y**

**}**

# Reference Parameters

- **When you use a reference parameter, the address (not the value) is automatically passed to the function**
- **A reference parameter is declared by preceding the parameter name in the function's declaration with and &**

## Reference Parameters

```
void f(int &i);
```

```
int main() {
```

```
    int val = 1;
```

```
    cout << "Old value for val: " << val << '\n';
```

```
    f(val); // pass address of val to f()
```

```
    cout << "New value for val: " << val << '\n';
```

```
    return 0;
```

```
}
```

```
void f(int &i) {
```

```
    i = 10; // this modifies calling argument
```

```
} /* declaration of i preceded by & causes it to  
    become a reference parameter*/
```

# Reference Parameters

**// Declare swap() using reference parameters.**

**void swap(int &x, int &y);**

**int main() {**

**int i, j;**

**i = 10;**

**j = 20;**

**cout << "initial values of i and j: ";**

**cout << i << ' ' << j << '\n';**

**swap(j, i);**

**cout << "swapped values of i and j: ";**

**cout << i << ' ' << j << '\n';**

**return 0;**

**}**

## Reference Parameters

**/\* Here, swap() is defined as using call-by-reference, not call-by-value. Thus, it can exchange the two arguments it is called with.**

**\*/**

```
void swap(int &x, int &y) {  
    int temp;  
    temp = x; // save the value at address x  
    x = y;    // put y into x  
    y = temp; // put x into y  
}
```

## Returning References

- **When a function returns a reference, it returns an implicit pointer to its return value**
- *The function can be used on the left side of an assignment statement*

# Returning References

```
double &f();
double val = 100.0;

int main()
{
    double newval;
    cout << f() << '\n'; // display val's value
    newval = f(); // assign value of val to newval
    cout << newval << '\n'; // display newval's value

    f() = 99.1; // change val's value
    cout << f() << '\n'; // display val's new value
    return 0;
}

double &f()
{
    return val; // return reference to val
}

```

## Returning References

- **Be careful that the return value being referred to does not go out of scope**

```
// Error, cannot return reference to local var.
```

```
int &f()  
{  
    int i=10;  
    return i;  
}
```

# Function Overloading

```
// Overload a function three times.  
void f(int i);    // integer parameter  
void f(int i, int j); // two integer parameters  
void f(double k); // one double parameter  
  
int main() {  
    f(10);    // call f(int)  
    f(10, 20); // call f(int, int)  
    f(12.23); // call f(double)  
    return 0;  
}
```

# Function Overloading

```
void f(int i)
```

```
{
```

```
    cout << "In f(int), i is " << i << '\n';
```

```
}
```

```
void f(int i, int j)
```

```
{
```

```
    cout << "In f(int, int), i is " << i;
```

```
    cout << ", j is " << j << '\n';
```

```
}
```

```
void f(double k)
```

```
{
```

```
    cout << "In f(double), k is " << k << '\n';
```

```
}
```

# Function Overloading

**// abs() is overloaded three ways.**

**int abs(int i);**

**double abs(double d);**

**long abs(long l);**

**int main()**

**{**

**cout << abs(-10) << "\n";**

**cout << abs(-11.0) << "\n";**

**cout << abs(-9L) << "\n";**

**return 0;**

**}**

# Function Overloading

```
int abs(int i) {  
    cout << "using integer abs()\n";  
    if(i<0) return -i;  
    else return i;  
}
```

```
double abs(double d) {  
    cout << "using double abs()\n";  
    if(d<0.0) return -d;  
    else return d;  
}
```

```
long abs(long l) {  
    cout << "using long abs()\n";  
    if(l<0) return -l;  
    else return l;  
}
```

## Default Function Arguments

```
void myfunc(double num = 0.0, char ch = 'X')  
{  
    .  
    .  
    .  
}
```

`myfunc(198.234, 'A');` // pass explicit values

`myfunc(10.1);` // pass num a value, let ch default

`myfunc();` // let both num and ch default

## Default Function Arguments

*// wrong!*

```
void f(int a = 1, int b);
```

```
int myfunc(float f, char *str, int i=10, int j);
```

## Default Function Arguments

```
void function(int i=1, int j=2, int k=3);
```

```
int main(void) {  
    function(1 1);  
    function(1 1,12);  
    function(1 1,12,13);  
  
    return 0;  
}
```

```
void function(int i, int j, int k) {  
    cout<<i<<"  "<<j<<"  "<<k<<endl;
```

## Default Function Arguments

```
void mystreat(char *s1, char *s2, int len = 0);
```

```
int main() {
```

```
    char str1[80] = "This is a test";
```

```
    char str2[80] = "0123456789";
```

```
    mystreat(str1, str2, 5); // concatenate 5 chars
```

```
    cout << str1 << '\n';
```

```
    strcpy(str1, "this is a test"); // reset str1
```

```
    mystreat(str1, str2); // concatenate entire string
```

```
    cout << str1 << '\n';
```

```
    return 0;
```

## Default Function Arguments

**// A custom version of strcat().**

```
void mystreat(char *s1, char *s2, int len) {  
    // find end of s1  
    while(*s1) s1++;  
    if(len==0) len = strlen(s2);  
    while(*s2 && len) {  
        *s1 = *s2; // copy chars  
        s1++;  
        s2++;  
        len--;  
    }  
    *s1 = '\0'; // null terminate s1  
}
```

## Overloading and Ambiguity

```
int myfunc(double d);
```

- 
- 
- 

```
cout << myfunc('c'); // not an error, conversion applied
```

# Overloading and Ambiguity

```
float myfunc(float i);
```

```
double myfunc(double i);
```

```
int main() {
```

```
// unambiguous, calls myfunc(double)
```

```
cout << myfunc(10.1) << " ";
```

```
// ambiguous
```

```
cout << myfunc(10);
```

```
return 0;
```

```
}
```

```
float myfunc(float i) {
```

```
    return i;
```

```
}
```

```
double myfunc(double i) {
```

```
    return -i;
```

```
}
```