


# C++ Functions

Like library functions, we can create our own functions

USE:

```
a = func1(b);
```



```
x = func2(y, 4.0) * 5.0;
```

```
r = func3(); // no arguments
```

Functions (usually) return a value back to the calling program

Function names follow same rules as variables

FUNC0010

# Function Definition

```
int funcname(int arg1, float arg2)
{
    // local variable declarations
    int val;

    // function body

    return val;
} // END func
```

*Note: main is a function, just like every other function*

FUNC0020

## The void Data Type

void is the type that indicates no type!

Example – no return value:

```
void funca(int a, char x)
{
    cout << "a is " << a << ", x is" << x << endl;
} // END funca
```

Example – no arguments:

```
float getPI(void)
{
    return 3.1415926;
} // END getPI
```

FUNC0030

## The Function Prototype

Every function must be known to the compiler before being used.

Serves same function as a variable declaration.

EX.

```
void funca(int, char);
int funcb(void);
```

- ▶ Prototype specifies the name of function and types of function and arguments.
- ▶ Names of argument variables don't need to be listed, ignored if specified.
- ▶ Note the semicolon at end of prototype! (No semicolon is allowed in the function definition!)

FUNC0040

## A Function Example

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

int dosum(int, int, int); // prototype

int main()
{
    int a, b, c, d;
    int x, y, z, w;
    cin >> a >> b >> c;
    cin >> x >> y >> z;
    d = dosum(a, b, c);
    w = dosum(x, y, z) * 3;
    cout >> d >> " " >> w >> endl;
} // END main
```

FUNC0050

## Example – the "dosum" function

```
int dosum(int r, int s, int t)
{
    int sum;

    sum = r + s + t;

    return sum;
} // END dosum
```

FUNC0060

## Revised Program Organization

```
#include<iostream>
// other includes go here

using namespace std;

// function prototypes go here

int main()
{
    //body of main program goes here

} // END main

// function definitions go here
```

FUNC0070

## Local Variables

```
int func(int, int);
int main()
{
    int a = 3, b = 4;
    int i, n;

    i = 0;
    n = func(a, b);
    cout << i << n << endl;
} // END main
```

```
int func(int x, int y)
{
    int i;

    i = x + y;

    return i;
} // END func
```

What happens with the variable i?

FUNC0080

# Variable Scope

The "scope" of a variable consists of those parts of the program where that variable is accessible.

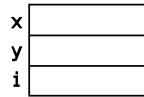
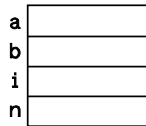
```
int main()
{
    int i, n, a, b;
    i = 0;

    n = func(a, b);
    cout << i << n << endl;
} // END main

int func(int x, int y)
{
    int i;

    i = x + y;

    return i;
} // END func
```



The scope of each *i* is the function where it is declared. It is called a "local" variable – separate memory is allocated for each *i*.

FUNC0090

# Global vs Local Variables

```
#include <iostream>
```

```
int a, b, c;
```

```
int main()
{
    int x, y, z;
    ⋮
} //END main
```

```
int func1 ()
{
    int r, s, t;
    ⋮
} //END func1
```

*a, b, c* are global

*x, y, z* are local to main

*r, s, t* are local to func1

FUNC0100

## Arguments – Call by Value

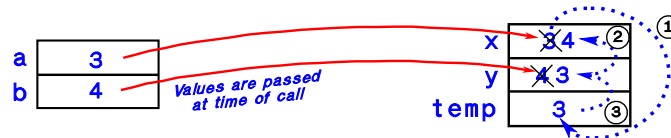
```

void swap(int, int);

int main ()
{
    int a = 3, b = 4;
    swap(a, b);
    cout << a << b << endl;
} //END main

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

```



*Only the copies of values are swapped!*

FUNC10

## Arguments – Call by Reference

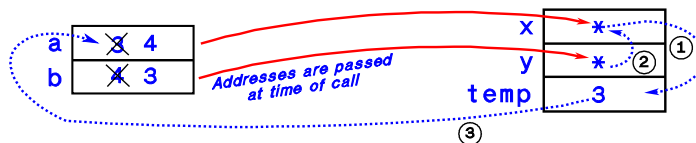
```

void swap(int&, int&);

int main ()
{
    int a = 3, b = 4;
    swap(a, b);
    cout << a << b << endl;
} //END main

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

```



FUNC120