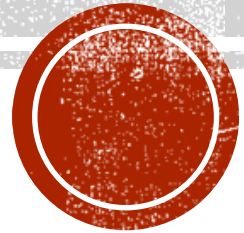
FUNCTIONS

Lab Instructor : Jean Lai



FUNCTIONS

- Group related statements to perform a specific task.
- Structure the program (No duplicate codes!)
- Must be declared before used.
- Can be invoked (called) as any number of times.
- Can call other functions.

- main() is also a function!

PREDEFINED FUNCTIONS

- C++ has many built-in functions that can be used.
- Must declare (#include) the libraries before using the functions.
- Used perform specific tasks.
- Some of the common functions include
  sqrt(), abs(), rand(), srand(), time() … etc.

double sqrt (double x);

- Returns the square root of x

```
#include <iostream>
#include <cmath>              /* required library for sqrt */
using namespace std;

int main () {
      double y = sqrt (25);      // y is the square root of 25
      cout << sqrt (45.9);       // print the square root of 45.9
      return 0;
}
```

int abs (int n);

- Returns the absolute value of n.

int rand ();

- Returns a pseudo-random integer in the range between 0 and RAND_MAX.

void srand (unsigned int seed);

- Initialize the pseudo-random number generator using a specific integer as the seed number.

```
#include <iostream>
#include <cstdlib>          /* required library for abs, rand, srand */
#include <time>             /* required library for time */
using namespace std;

int main () {
      int y = abs(-25);          // y is the absolute value of -25
      cout << abs(49);           // print the absolute value of 49

      srand (time (NULL));       // using current time as the seed number
      int v1 = rand() % 100;           // v1 in the range 0 to 99
      int v2 = rand() % 100 + 1;       // v2 in the range 1 to 100
      int v3 = rand() % 30 + 1985;     // v3 in the range 1985-2014
      return 0;
}
```

DEFINE THE FUNCTION

```
return_type function_name (argument_list) {
  //function body
  //do something
}
```

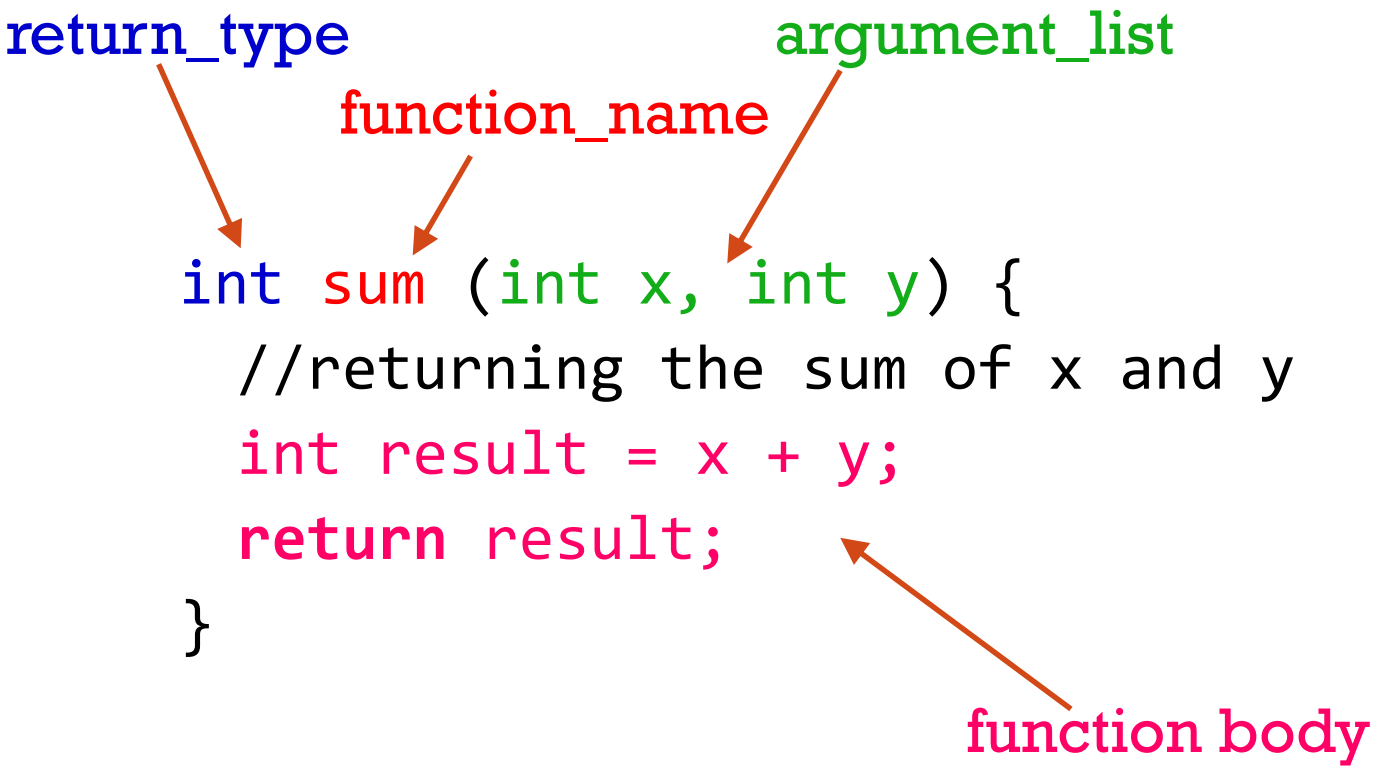
return type – what kind of value the function returns, can be any of C++ data types: int, double, bool, char, string, …etc. or void if the function does not return a value.

function_name – name of the function similar to variable names.

argument_list – a list of comma separated arguments (also known as parameters)

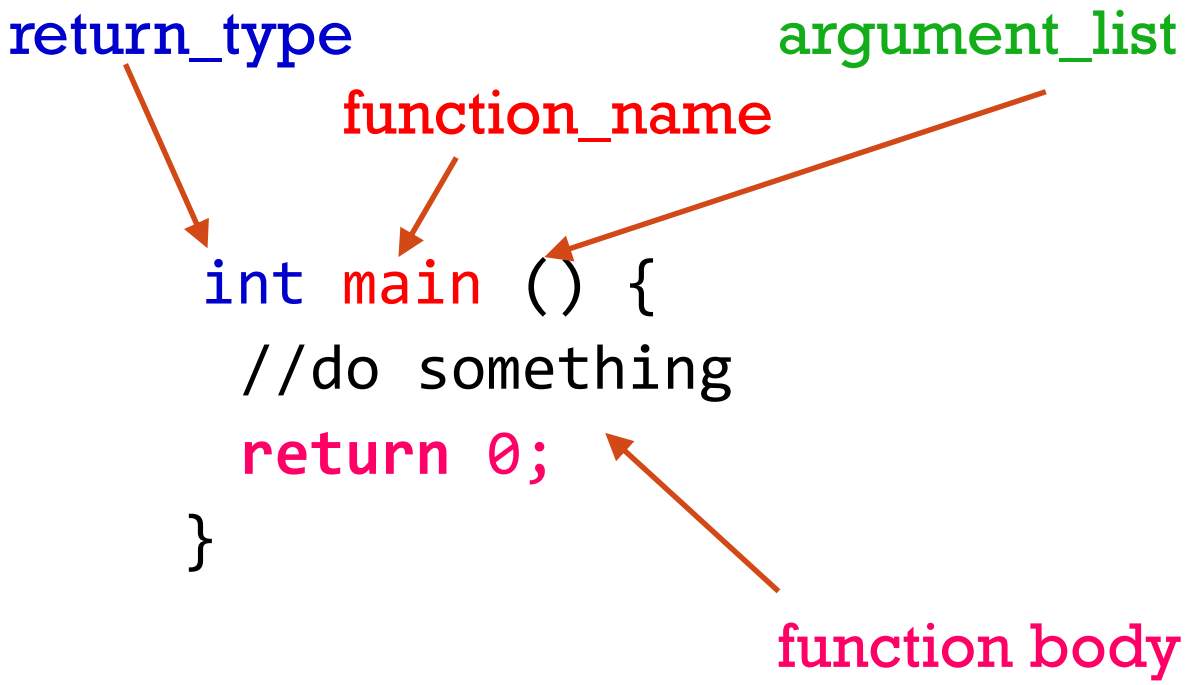
function body – list of statements accomplishing a task.

EXAMPLE



❖**NOTE**: Use keyword `return` to exit the function. Must return the value of same return type.

FUNCTION MAIN( )



MORE FUNCTION HEADINGS

```
double avg (double x, double y) {
      return (x + y) / 2;
}

bool isPrime (int x) {
      …
      return true;
}
```

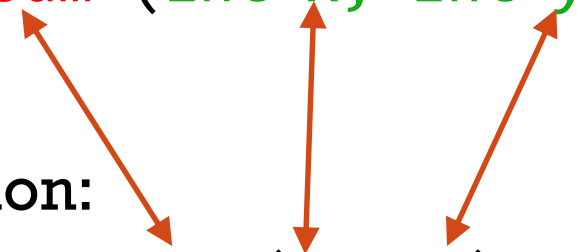
FUNCTION CALL

- To execute the function.
- Argument types must match argument_list in the function heading.
- Is an expression.
- Can be used within a larger expression if the return type is not void.

CALL THE FUNCTION

- Function heading:

```
int sum (int x, int y) { … }
```



- Calling the function:

```
int result = sum (8, 17);
```

or

```
cout << “The sum is ” << sum(8, 17);
```

MORE FUNCTION CALLS

```
bool isPrime (int x) { … }

if (isPrime(7))
      cout << “7 is prime.”;
```

```
void printHello (int x) { … }

int x = 8;
printHello(x);
```

SAMPLE CODE

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

int sum(int x, int y) {
	return x + y;
}

int main() {
	int x, y;
	cin >> x, y;
	cout << x << “ + ” << y << “ = ” << sum(x, y);
	return 0;
}
```

SAMPLE CODE (WITH FUNCTION DECLARATION BEFORE MAIN)

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

int sum(int x, int y);
int main() {
	int x, y;
	cin >> x, y;
	cout << x << " + " << y << " = " << sum(x, y);
	return 0;
}

int sum(int x, int y) {
	return x + y;
}
```

SCOPE OF VARIABLE

- Extent of visibility of the variable.
- Exist only inside the block (**curly braces { }**) where it is declared.
- Can be either local or global scope.

SCOPE EXAMPLE

```
int sum (int x, int y) {
    int z = x + y;
    return z;
}
```

local variable z, not accessible outside the function!

local variable r, accessible to all within its block { }

```
for (int r = 1; r <= 5; r++ ) {
        for (int c = 1; c <= r; c++) {
                cout << "*";
        }
        cout << endl;
}
```

local variable c, only accessible within inner loop!

SCOPE EXAMPLE

```
int main  () {
int n = 8;

for (int r = 1; r <= n; r++ ) {
        for (int c = 1;  c <= n; c++) {
                cout << "*";
        }
        cout << endl;
}
return 0;
}
```

local variable n, visible to all nested within the same block { }!

PASS BY VALUE

- A **copy** of the original value is passed into the function.
- Original value is not altered.
- May use different variable names but must have same data type.
- Seen as a local variable in the function.
- Once function exits, the local variable is discarded.

PASS BY VALUE EXAMPLE

```
void changeN(int n) {
  n = n + 7;
}


int main  () {
  int n = 8;
  cout << "The original value of n : " << n;
  changeN(n);
  cout << "The updated value of n : " << n;

  return 0;
}
```

The value of **n** *is not changed by the call to* **changeN**.

PASS BY REFERENCE

- The original variable is passed into the function.
- Original value will be altered.
- May use different variable names but must have same data type.
- Must attach the ampersand sign & to the parameters in argument list.
- Once function exits, the changes still remain!

PASS BY REFERENCE EXAMPLE

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


int main  () {
  int x = 2, y = 5
  cout << "Original x : " << x << " y : " << y;
  swap(x, y);
  cout << "Updated x : " << x << " y : " << y;

  return 0;
}
```

The value of **x** and **y** *is changed by the call to* **swap**.

- C++ allows ampersand associated either with type name or parameter.

```
void swap(int& x, int& y);
          is the same as
void swap(int &x, int &y);
```