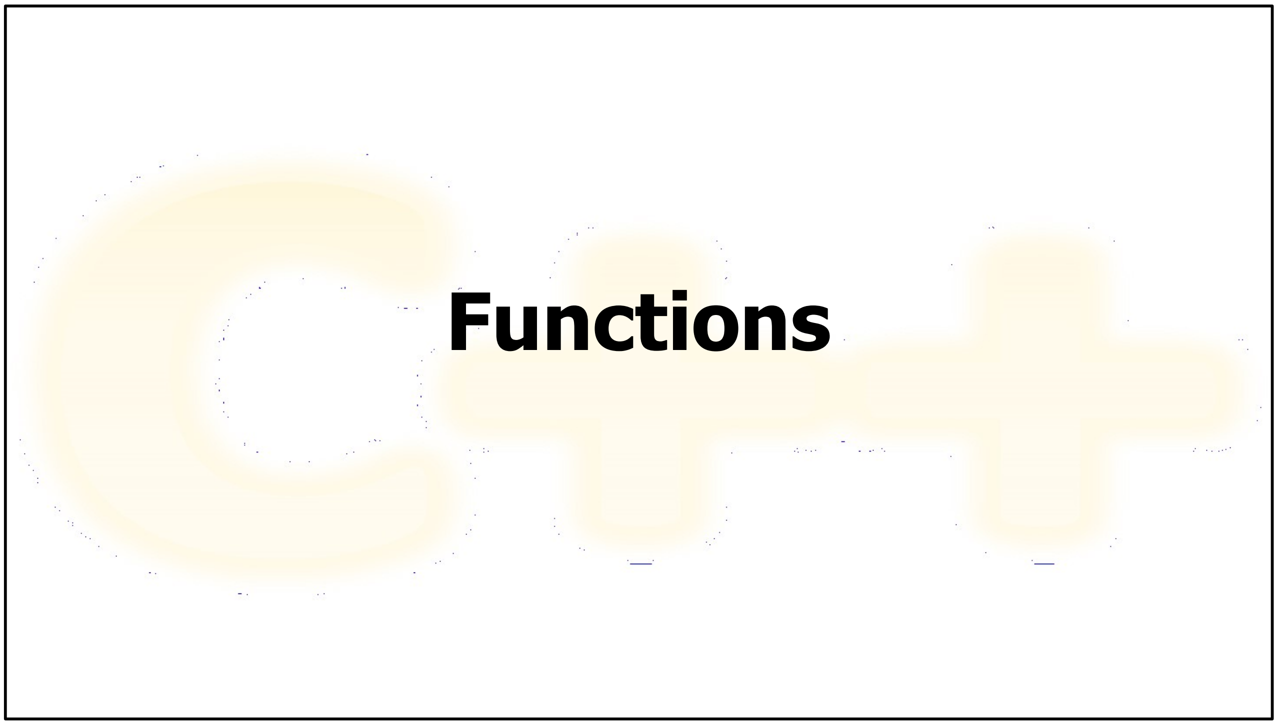
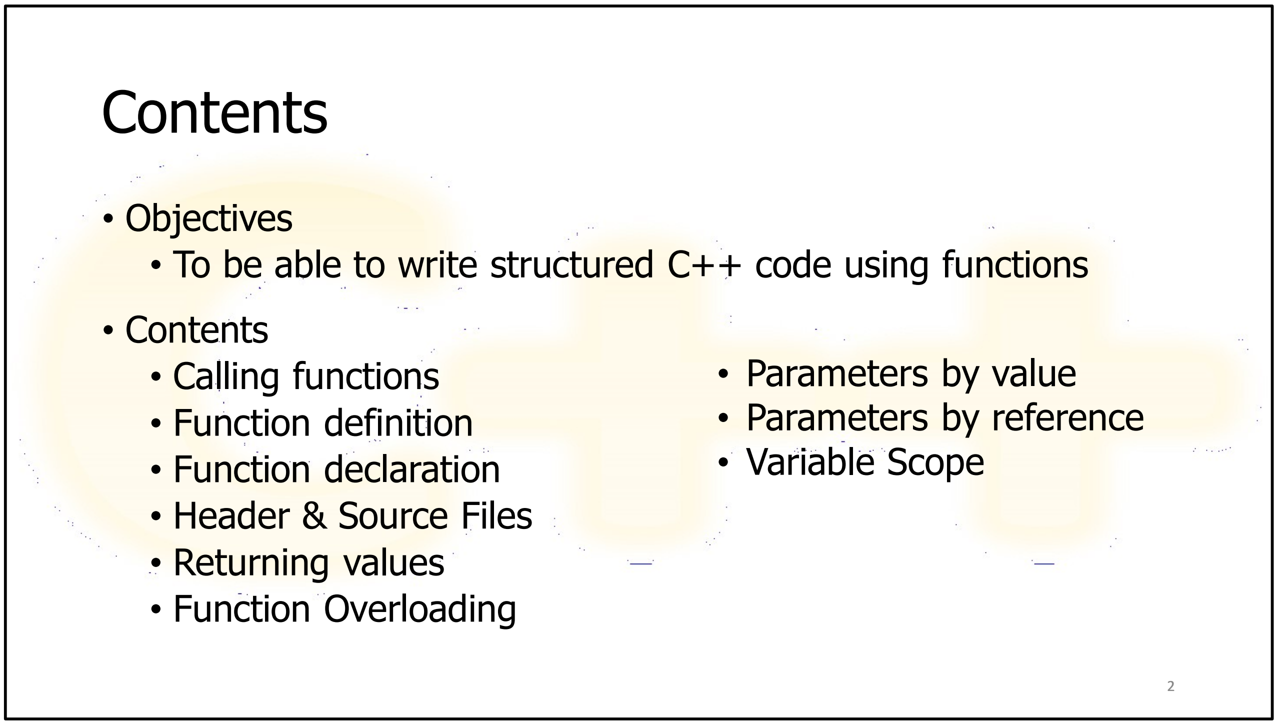


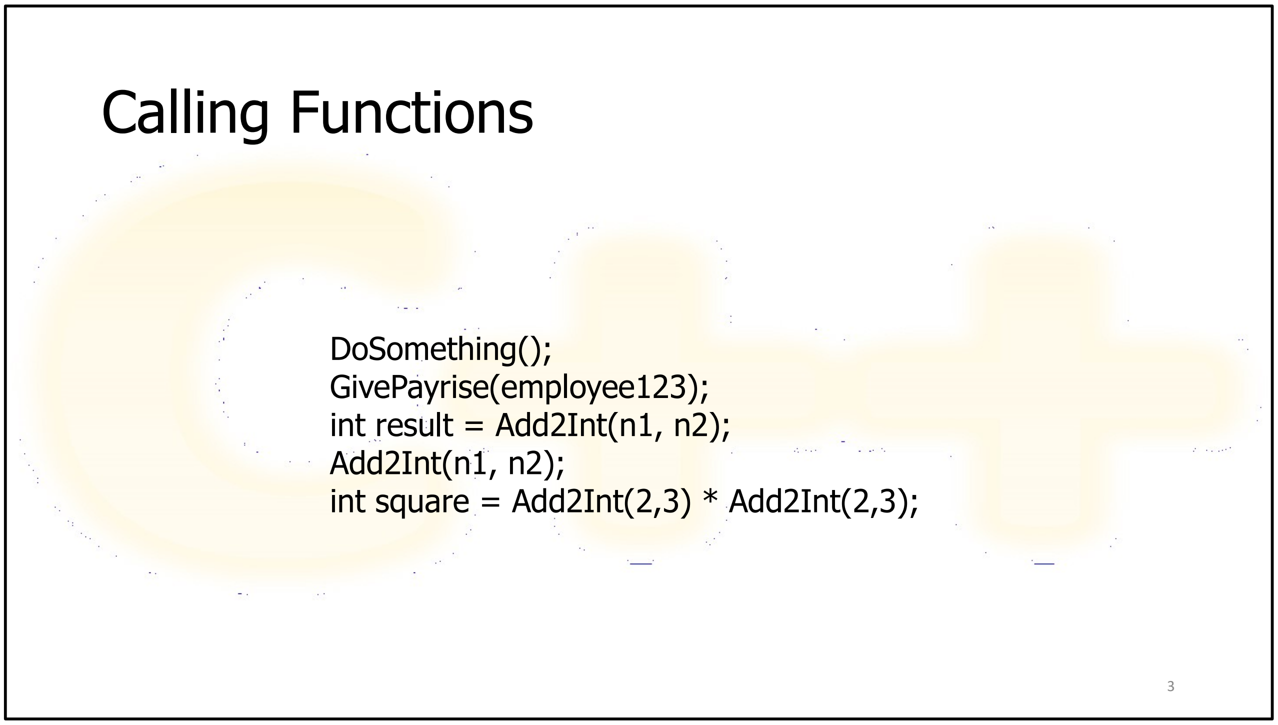
Functions



Contents

- Objectives
  - To be able to write structured C++ code using functions
- Contents
  - Calling functions
  - Function definition
  - Function declaration
  - Header & Source Files
  - Returning values
  - Function Overloading
  - Parameters by value
  - Parameters by reference
  - Variable Scope

2



Calling Functions

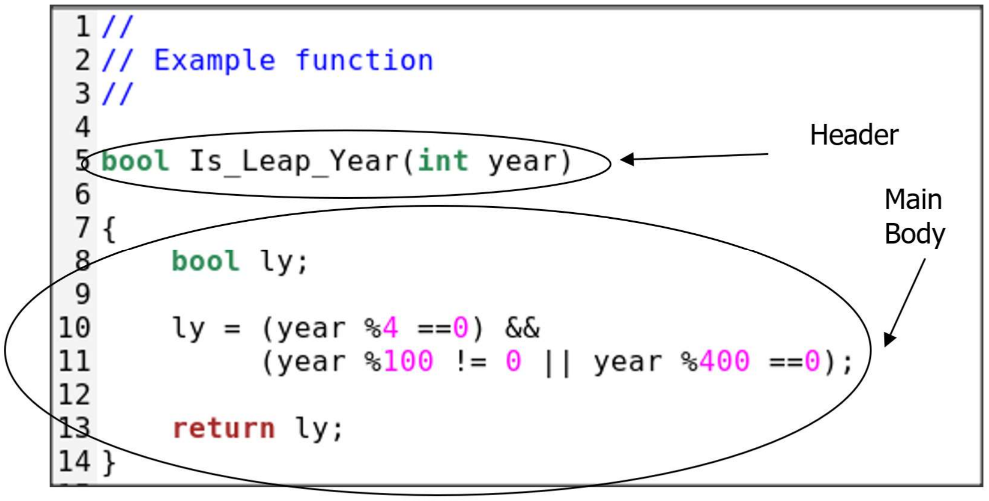
```
DoSomething();
GivePayrise(employee123);
int result = Add2Int(n1, n2);
Add2Int(n1, n2);
int square = Add2Int(2,3) * Add2Int(2,3);
```

3

A C++ function is called by specifying its name followed by a list of arguments enclosed in parentheses. Values can be returned either through the arguments or through the function itself. For now we will only consider passing values back through the function call. Returning values through arguments will be discussed later in this section.

If a result is passed back through the function the function call can then be used in an expression.

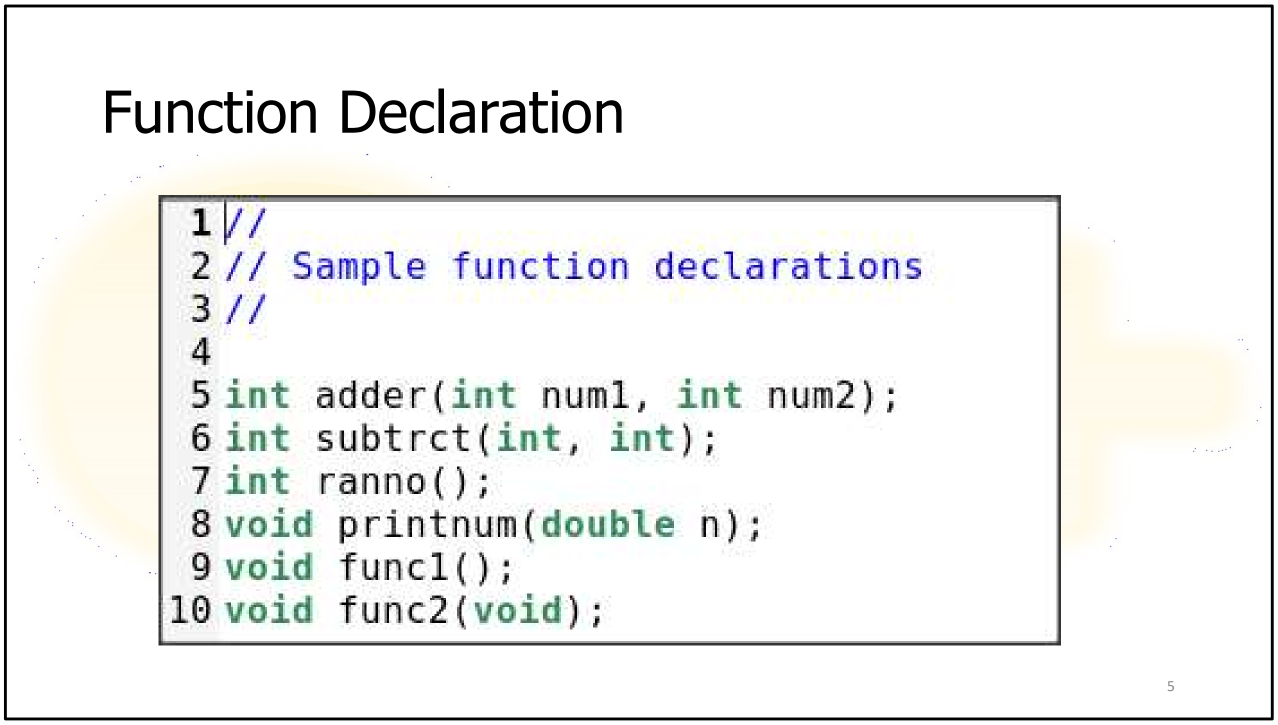
Function Definition



```
//
// Example function
//

bool Is_Leap_Year(int year)

{
    bool ly;

    ly = (year %4 ==0) &&
         (year %100 != 0 || year %400 ==0);

    return ly;
}
```

A function definition consists of a header and a main body. The function header is identical to the function prototype / declaration except that it has no trailing semicolon.

The main body of the function is a block of code. This is the code that is executed when the function is called.



Function Declaration

```
//
// Sample function declarations
//

int adder(int num1, int num2);
int subtrct(int, int);
int ranno();
void printnum(double n);
void func1();
void func2(void);
```

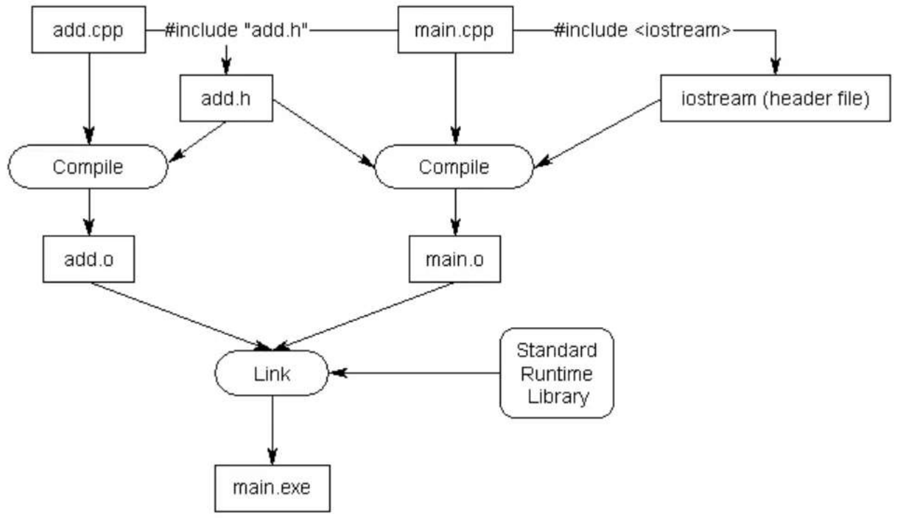
5

Function declarations are also called prototypes. The prototype for a function lists the return type for the function and the arguments used.

A function can return any type of object except an array.

Although it is acceptable to not code argument names in a function prototype it is good programming practice to code them with meaningful names. These must be valid C++ identifier names.

Header & Source Files



6

Header files contain function declarations among other things. In C++ terminology a function declaration is called a prototype. Also declared in header files are any class / structure definitions or typedefs that are used within the source module.

Because a header file might potentially be included by multiple files, it cannot contain definitions that might produce multiple definitions of the same name. The following are not allowed, or are considered very bad practice:
•built-in type definitions at namespace or global scope
•non-inline function definitions
•non-const variable definitions
•aggregate definitions
•unnamed namespaces
•using directives

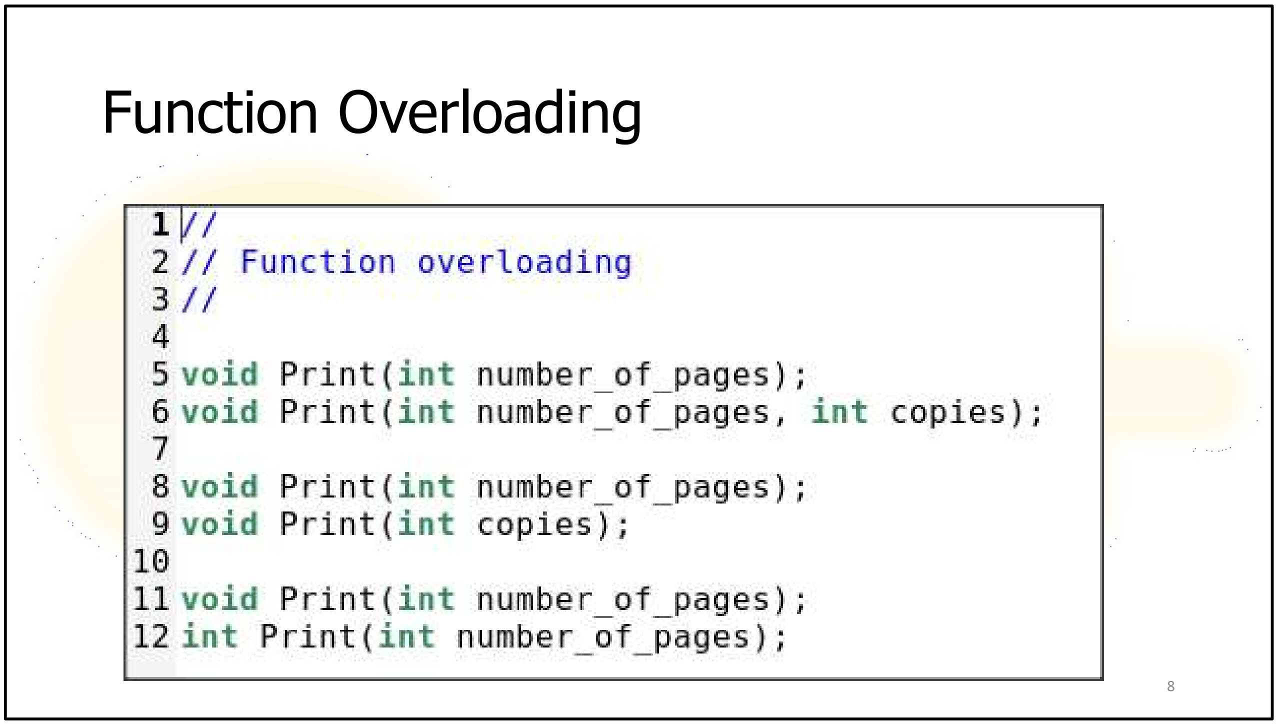
Header files are included into source files with the #include compiler directive.

The extension is usually .h or .hpp

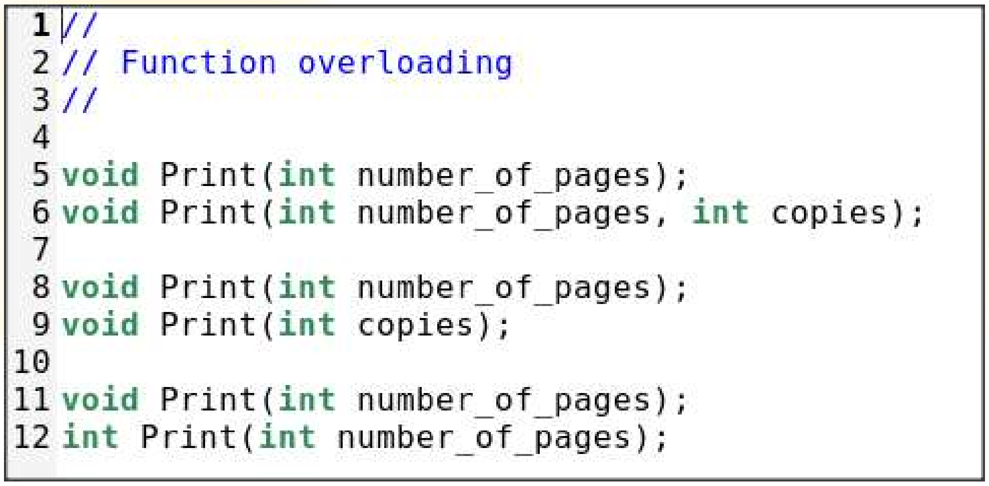
There are some predefined or system header files. These include files such as vector, string, iostream, math.h etc. To include system header files the filename should be enclosed in angle brackets. For user header files the filename should be enclosed in double quotes.

Returning Values

- A function can return a single value back to the calling code through the return statement.
- The type of the value should be the same type as the function definition in the prototype.
- Execution of the function will terminate when the first return statement is encountered.
- In a void function the return statement may be coded with no return value.
- Alternatively the entire return statement may be omitted.
- In this case execution of the function ends when the end of the main body of the function is reached.



Function Overloading



```cpp
//
// Function overloading
//

void Print(int number_of_pages);
void Print(int number_of_pages, int copies);

void Print(int number_of_pages);
void Print(int copies);

void Print(int number_of_pages);
int Print(int number_of_pages);
```

8

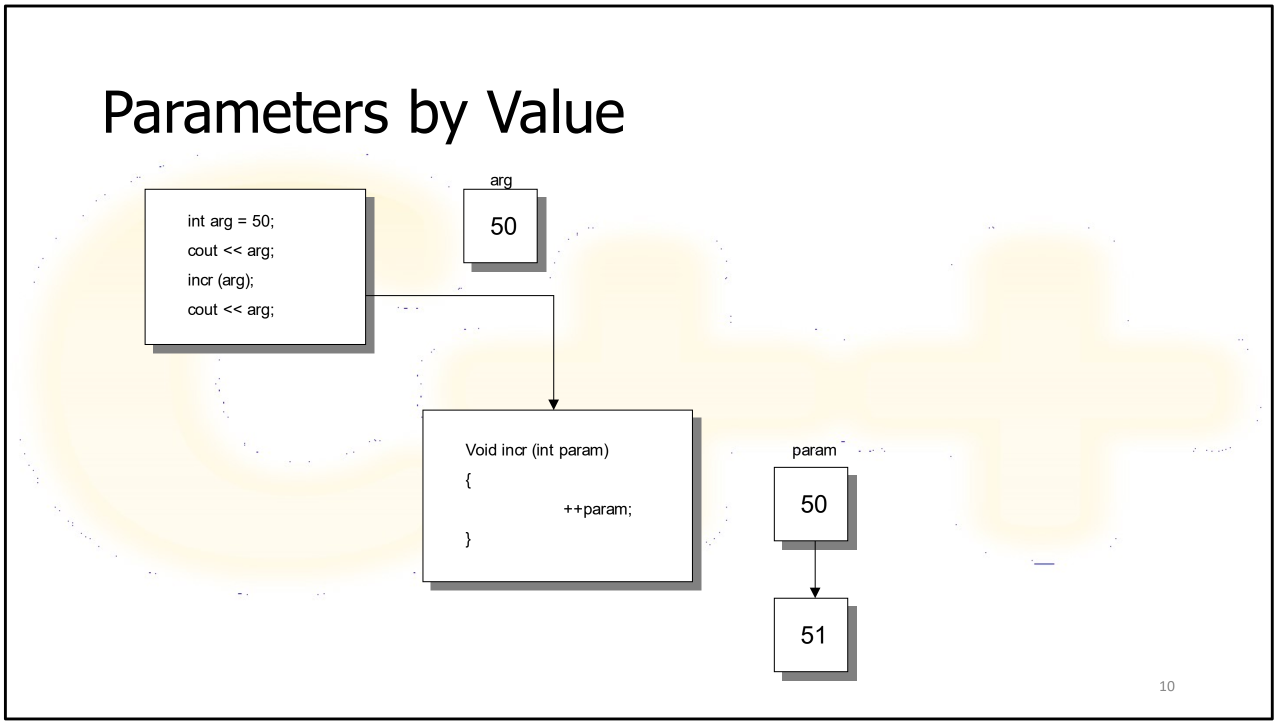
C++ allows several functions to share the same name. The functions MUST have different argument lists based on the number and types of the arguments. The different functions are identified by their signature. A function signature consists of its name and argument list.

The compiler 'mangles' the function name to create a unique name for each function that includes information about the parameter types (among other things). The name mangling is compiler dependent. (Simple in g++ more complex in Microsoft compilers).

Function Overloading

9

Write a number of valid / invalid overloads on whiteboard. Delegates have to state whether they are valid or not & if not why not! Valid overloads are left on board & the next delegate must check their overload against 'all the above'.



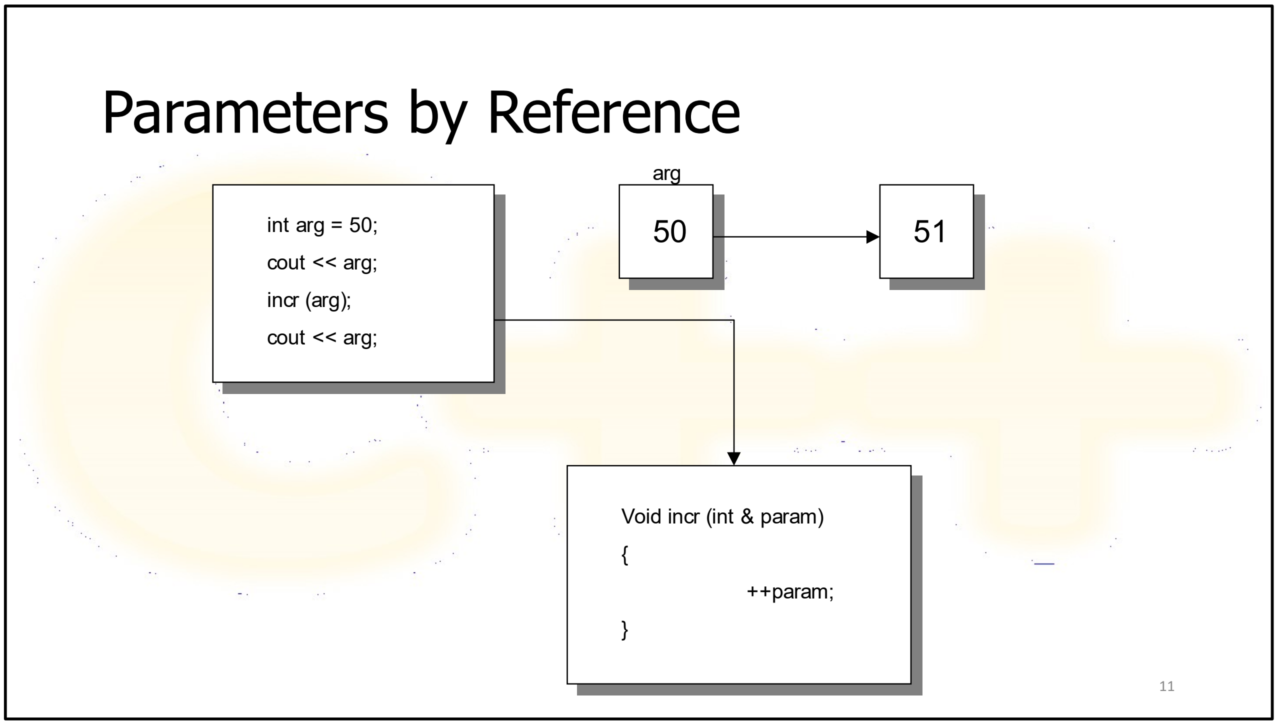
Parameters by Value

```
int arg = 50;
cout << arg;
incr (arg);
cout << arg;
```

```
Void incr (int param)
{
        ++param;
}
```

Passing by value is the most common way of passing values into a function. It is also the default method. Passing by value is sometimes called passing by copy because when using this method of passing values a copy is made of the original argument value. The copy is local to the called function. The copy is performed in one direction only. I.e. the copy is only performed at the start of the called function. Any modified data is never copied back to the calling code.

Arg is declared as an integer with an initial value of 50. The first cout will print 50. This is correct. The incr function is then called passing arg by value or by copy. A local copy of arg is made within the incr function called param. param is then incremented. When incr is exited the local param is destroyed leaving arg with it’s original value. The second cout statement will print arg as 50.



Parameters by Reference

```
int arg = 50;
cout << arg;
incr (arg);
cout << arg;
```

```
Void incr (int & param)
{
        ++param;
}
```

11

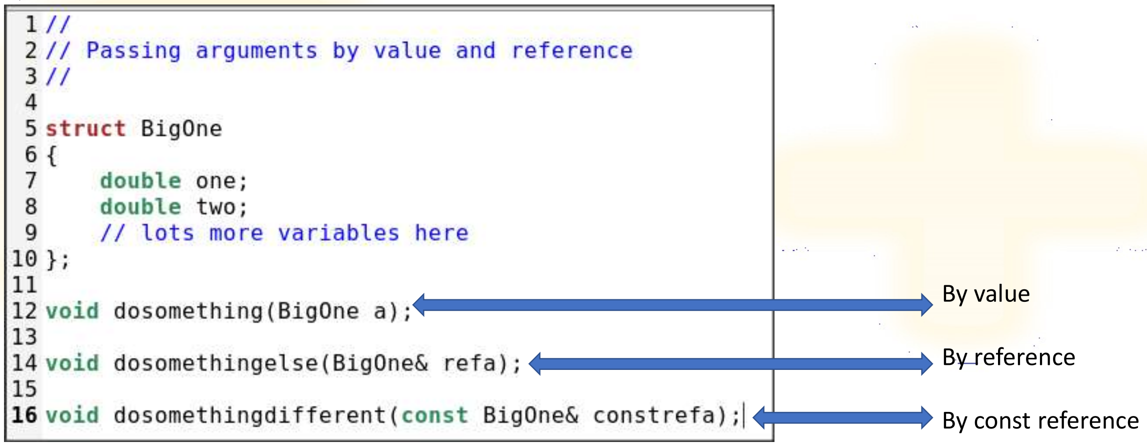
When parameters are passed by reference, effectively, the address of the original variable is passed into the called function. No copying of parameters takes place. For this reason, pass by reference is generally more efficient.

In this example the function declaration has changed. The argument is declared as int & param. & is used to pass a synonym of the original variable to the function. When param is incremented it is the original variable that is incremented. No copying has taken place and the arg variable has been changed rather than a copy of arg.

In this example the first cout statement would print 50 as before. However the second cout statement would print 51.

It is not permissible to pass literal values into a call by reference argument.

Parameter Types

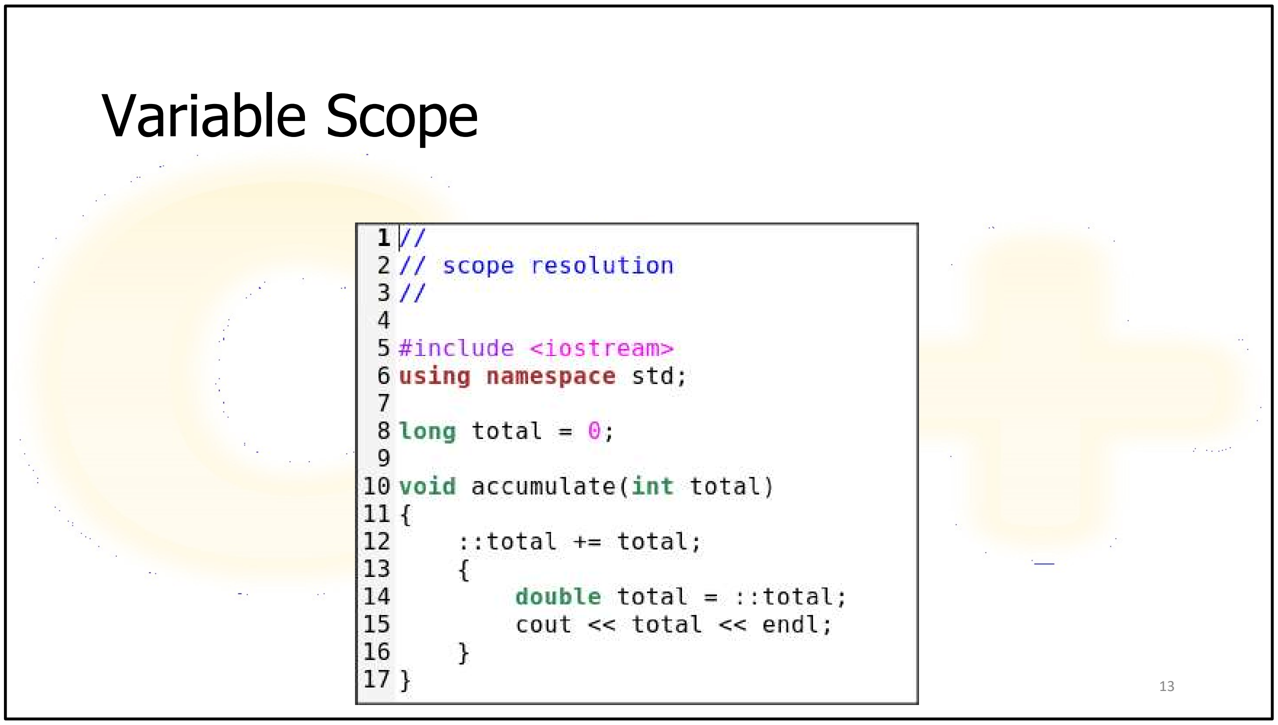


```
//
// Passing arguments by value and reference
//

struct BigOne
{
    double one;
    double two;
    // lots more variables here
};

void dosomething(BigOne a);

void dosomethingelse(BigOne& refa);

void dosomethingdifferent(const BigOne& constrefa);
```

12

Built in types are part of the C++ language. They are permanently small and, thus, trivial to copy. They can be used for arguments passed by value. Copying of built in types is implemented efficiently by the C++ compiler.

Complex data types (generally those which require a #include to implement them) are usually larger than the built in types. They have the potential to be very large, and thus, non-trivial to copy. They should not, therefore, be passed by value. User defined types should always be passed by reference. If a read only value is required then it can be passed as const by reference.



Variable Scope

```
//
// scope resolution
//

#include <iostream>
using namespace std;

long total = 0;

void accumulate(int total)
{
    ::total += total;
    {
        double total = ::total;
        cout << total << endl;
    }
}
```

13

The scope of a variable defines where the variable can be referenced. Variables defined within a function have local scope. They can only be referenced from within the function in which they are declared.

Variables declared as part of a for loop are local to the loop and can only be referenced from the block of code that is controlled by the loop.

Variables declared outside functions either in the source modules or in the header files have global scope. They can be referenced from anywhere.

Exercise

Write a function that takes an integer passed by value and returns the square and cube of the number as reference parameters. In the main program read a number from the keyboard, call the function & print the number & its square and cube.