Introduction to Visual Basic and Visual C++

Lesson 9

Introduction to C++

A Quick Look to C++

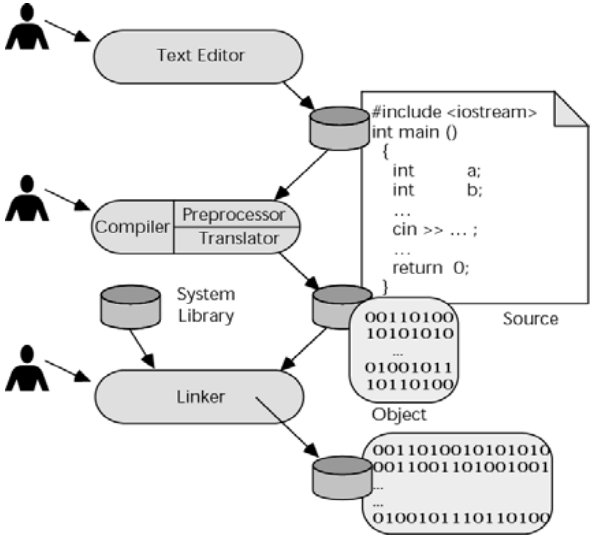
Introduction

- C++ is a structured programming language – by design.
- That is, it is meant to facilitate the writing of programs using small logic modules that perform a single function and have both a one entry point and a one exit point.
- C++, like almost all programming languages, is algorithmic. Meaning it is a step by step set of instructions for solving a problem or completing some task.
- The first common block structured programming language was ALGOL for **ALGO**rithmic **L**anguage.

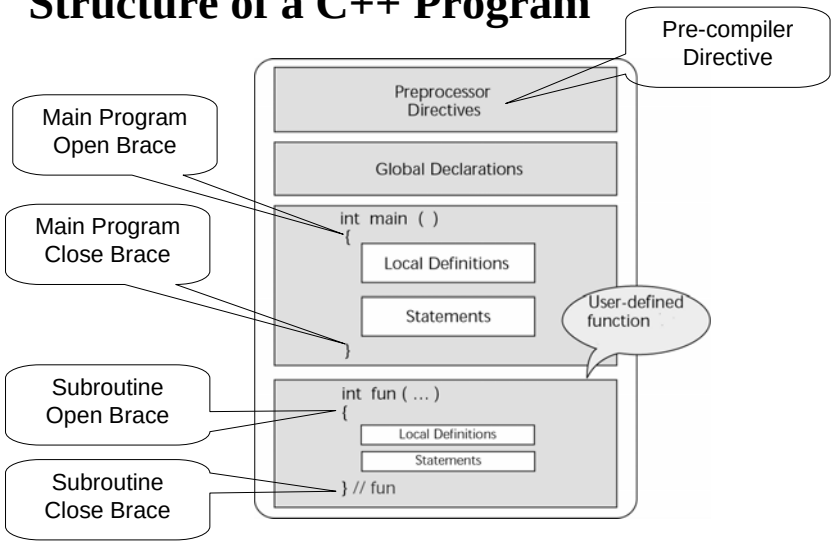
What is C++?

- C++ is a high level programming language.
- In either machine language or assembler (symbolic language) each line of code results in a single instruction.
- In a high level language a simple instruction like:
  - result = value1 + (value2 * tax_rate);
- will cause many lines of code to be generated. This minimizes the chance of coding errors. The down side is that the programmer gives up absolute control over the generated code.

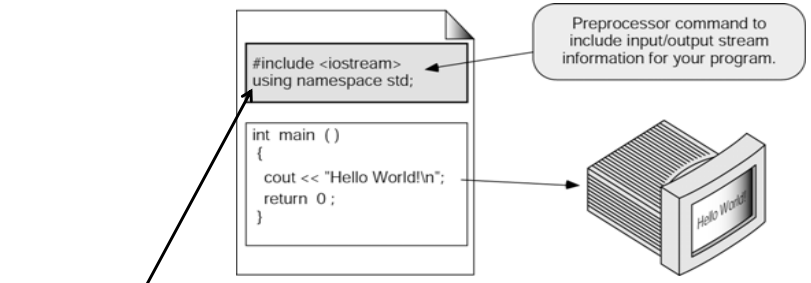
Building a C++ Program



Structure of a C++ Program



Preprocessor Directive



Namespace std contains all the classes, objects and functions of the standard C++ library.

Without namespace

```
#include <iostream>
int main () {
 std::cout << "Hello world!\n";
  return 0;
 }
```


Preprocessor Directive

- It is denoted by **#include <iostream>**
  - **#include** is used to call/load header file
  - **iostream** is a header file, it is required when input & output functions are used in programs
- Other commonly used header files:
  - cstdlib
  - ctime
  - cmath
- Self defined header file is written as:
  - **#include "***.h**"**, whereby " * " is the file name
  - E.g. **#include "myHeader.h"**

Headers

- Older compilers do not recognized some of the header files without the .h at the end. E.g. Turbo C++ 4.5 compiler does not recognized **iostream** but will recognize **iostream.h**

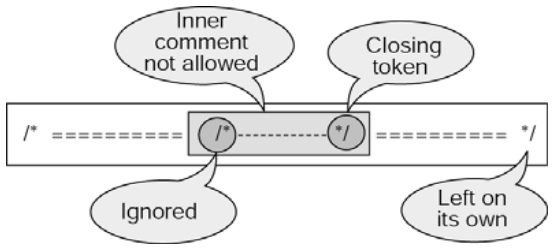
algorithm	cstdlib	iostream	set
bitset	cstring	iterator	sstream
cassert	ctime	limits	stack
cctype	deque	list	stdexcept
cfloat	fstream	locale	string
climits	functional	map	typeinfo
cmath	exception	memory	utility
cstdio	iomanip	queue	vector

Namespaces

- A namespace is somewhat like the name (or number) of a chapter in a book
- All the files in the C++ standard library declare all of its entities within the std namespace
- C++ has the standard namespace which is abbreviated **std**, **cout** and **cin** are within the standard namespace
- Brings the whole of the standard template library (STL) into the current scope of the program
  - **`using namespace std;`**
- To bring a single namespace member into scope, the using keyword can be used to explicitly refer to that item
  - **`using std::cout;`**
  - **`using std::cin;`**
  - **`using std::endl;`**

Comments

- It is used to describe program and good for self-documentation
- Examples:
  - `// This is a single line comment`
  - `/* This is a`
    `multiline comment */`



`main` Function

- The body of the program
  - **`int main()`**
    - **`int`** is a return-value-type of integer value number
    - **`main`** is the function name, and is compulsory
    - **`()`** is known as parenthesis. Used in function to get parameter list
- **`{`** is called left brace, **`}`** is called right brace
  - It is required to indicate the beginning and ending of a function
  - It is also to indicate a block

Ending Function

- Some programs end with the following line
  - **`return 0;`**
    - A way to exit a function
    - **`return 0`** in this case, means to terminate the program normally
- If you self-define some functions, the `return` value varies
- Some programs do not have the return keyword because of the `return-value-type` of `void`
- The right brace **`}`** must be used to indicate the end of a main function

First C++ Program

```
/* A program that adds up two numbers */
#include <iostream> /* preprocessor directive */
using namespace std; /* allows us to easily access C++ built-in resources
   such as input and output */

int main() /* main function */
{ /* main function begins */
   int num1, num2, total; /* declare variable names */

   cout << "Enter two integers:" << endl; // prompt user for input
   cin >> num1 >> num2; /* getting input from user */

   total = num1 + num2; /* calculation statement */
   cout << "\nThe total is: " << total << endl; // display output
   system("pause");
   return 0;
} /* main function ends */
```

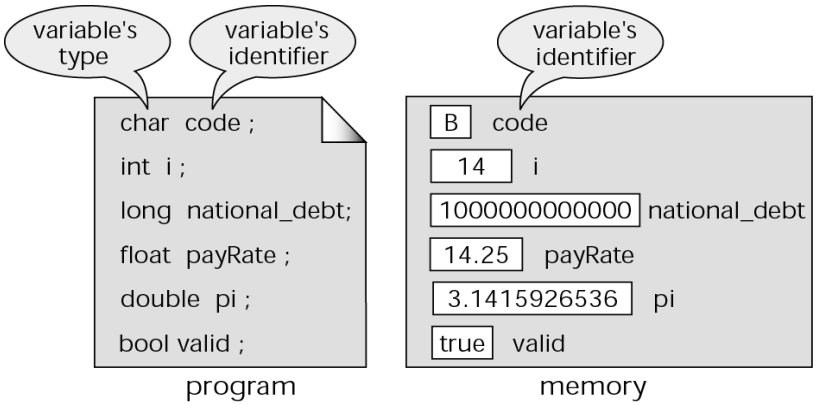

Identifiers

Variables and Data Type

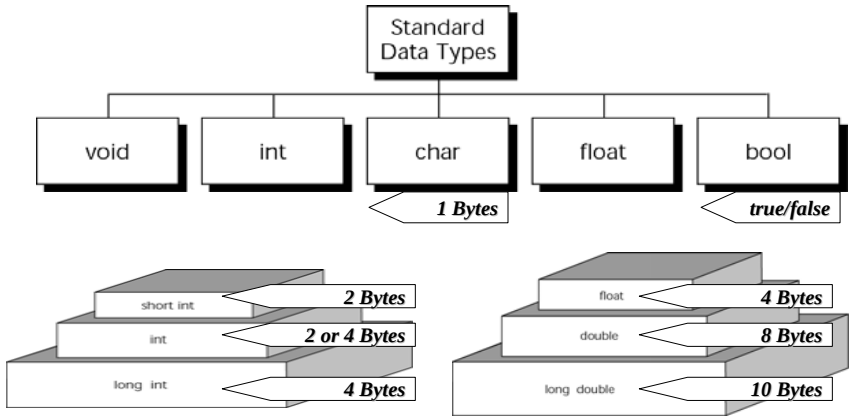
Identifiers

- Identifiers are also known as labels. They are the feature that allow us to refer to data or other objects by name.
- The rules for identifiers are:
- The first character must be either an alphabetic or an underscore character;
- The identifier must consist only of alphabetic characters, digits, and underscores;
- The identifier may not duplicate a reserved word.
- Good identifiers are both descriptive and short.

Variables Declaration

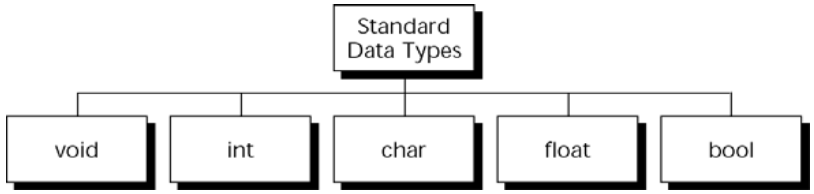


Standard Data Types



Size of value type can be different by Computer/OS types

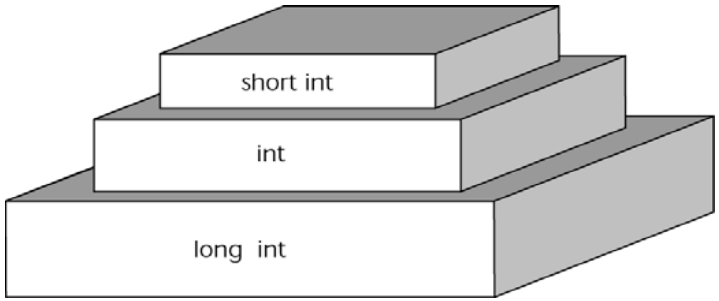
Standard Data Types



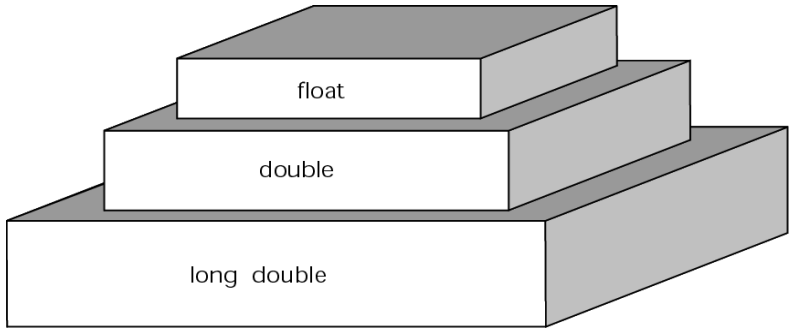
Type Name	Bytes	Other Names	Range of Values
int	*	signed, signed int	System dependent
unsigned int	*	unsigned	System dependent
__int8	1	char, signed char	-128 to 127
__int16	2	short, short int, signed short int	-32,768 to 32,767
__int32	4	signed, signed int	-2,147,483,648 to 2,147,483,647
__int64	8	none	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
bool	1	none	false or true
char	1	signed char	-128 to 127
unsigned char	1	none	0 to 255
short	2	short int, signed short int	-32,768 to 32,767
unsigned short	2	unsigned short int	0 to 65,535
long	4	long int, signed long int	-2,147,483,648 to 2,147,483,647
long long	8	none (but equivalent to __int64)	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
unsigned long	4	unsigned long int	0 to 4,294,967,295
enum	*	none	Same as int
float	4	none	3.4E +/- 38 (7 digits)
double	8	none	1.7E +/- 308 (15 digits)
long double	same as double	none	same as double
wchar_t	2	__wchar_t	0 to 65,535

Integer Types

- C++ dictates that the number of bits in a “short” is less than or equal to the number of bits in an “int” which is less than or equal to the number of bits in a “long”. The actual number varies by both system and compiler. This can be a real problem is the same code is going to be used on many differing types of systems.

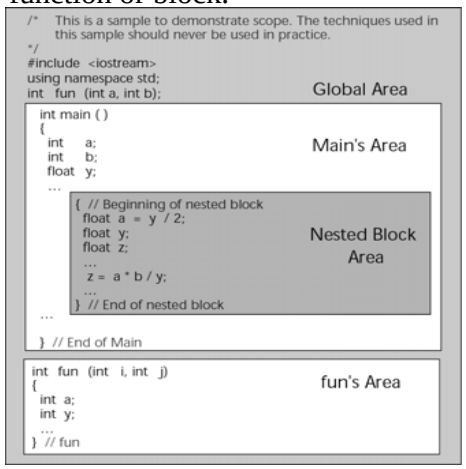


Floating-point Types



Scope for Global and Block Areas

- Variables are in scope from their point of definition until the end of their function or block.

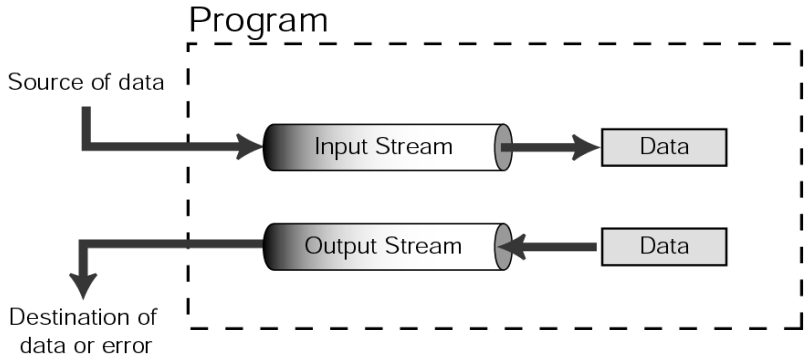


I/O Streams

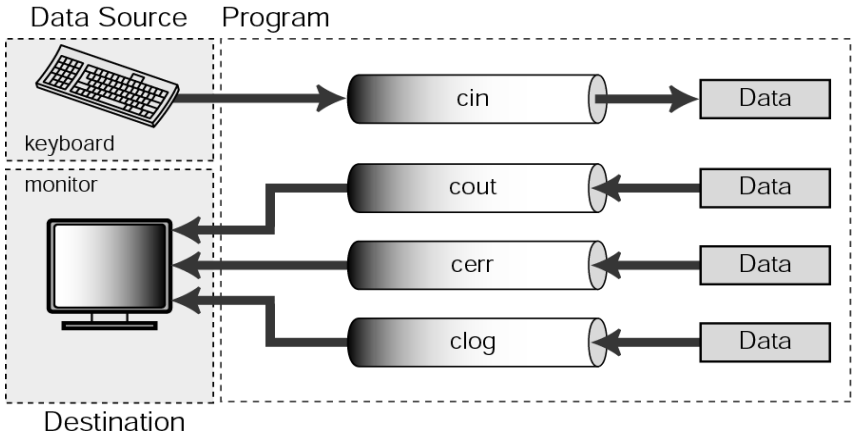
Input and Output Entities

Streams Conspet

- Standard streams are created, connected, and disconnected automatically.



Standard I/O Streams



cin Statement

- It is used to get input from user, usually keyboard
- **cin** is an object that is similar to cout
- A cout statement immediately before a cin statement is used to indicate to user what to enter at the keyboard
- This cout statement is called a prompt
  - **cin** >> *identifier1* >> *identifier2* >> **… ;**
- The >> symbol is called the extraction operator / input stream operator
- When the cursor is blinking on the screen, it indicates that user has to input some numbers/text and then press the Enter (Return) key as a response back to the program

cout Statement

- It is used to print/display out characters, text and numbers. Any characters, text, numbers in between double quotes will be printed out, except escape characters
- An object is a region of storage in memory, and in C++ programs, the **cout** region of memory is linked to the stand output device
- All statements must end with a semicolon (;), also known as statement terminator
  - **cout << "Enter two integers:" << endl;**
    - << is called the insertion operator / output stream operator, and it is used in cout statement
    - **endl** is used to create a new line, in other words, it places the insertion cursor at the beginning of the next line

cout Properties

- Setting Output Width
  - You can use the **width(**int**)** function to set the width for printing a value, but it only works for the next insertion command:
- Setting the Fill Character
  - Use the **fill(**char**)** function to set the fill character. The character remains as the fill character until set again.
- Significant Digits in Float
  - Use function **precision(**int**)** to set the number of significant digits printed (may convert from fixed to scientific to print)

cout Properties (Cont')

- Output Manipulators
  - Manipulators included like arguments in extraction
    - endl - outputs a new line character, flushes output
    - dec - sets int output to decimal
    - hex - sets int output to hexadecimal
    - oct - sets int output to octal

Example

```
int x = 42;
cout.width(5);
cout.fill('*');
cout << x << '\n';  // Outputs ***42

float y = 23.1415;
cout.precision(3);
cout << y << '\n'; // Outputs 23.1

cout << oct << x << endl; // Outputs 52\n
cout << hex << x << endl; // Outputs 2a\n
cout << dec << x << endl; // Outputs 42\n
```


Escape Characters

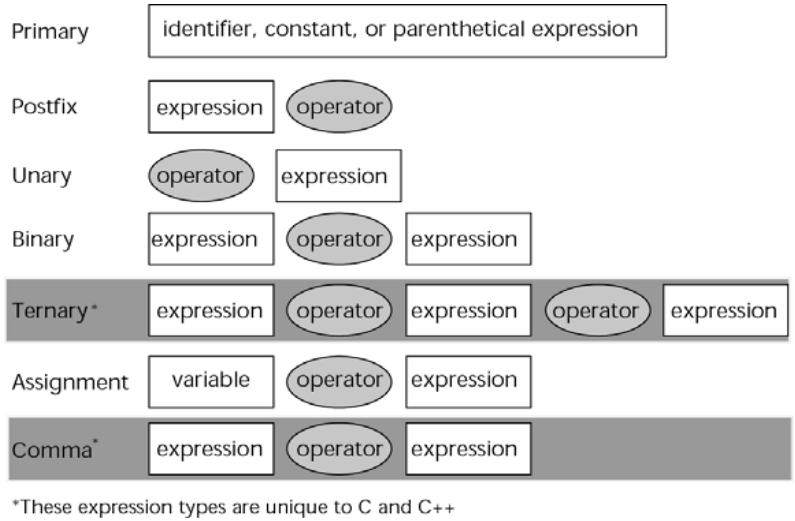
- Escape character (\)
  - Indicates that **`cout`** should do something out of the ordinary

Escape Sequence	Meaning	Description
`\n`	Newline	Position the cursor at the beginning of the next line
`\t`	Horizontal tab	Move the cursor to the next tab stop
`\\`	Backslash	Displays the \ to the screen
`\r`	Carriage return	Moves the active position to the initial position of the current line
`\b`	Backspace	Moves the active position back one space on the current line and deletes one the previous character
`\"`	Double quote	Displays the " to the screen
`\0`	Null character	Terminates a character string

Expressions

Introduction to C++ Expression

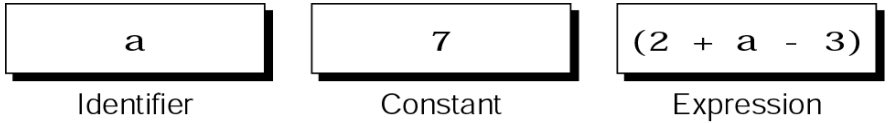
C++ Expression Format



*These expression types are unique to C and C++

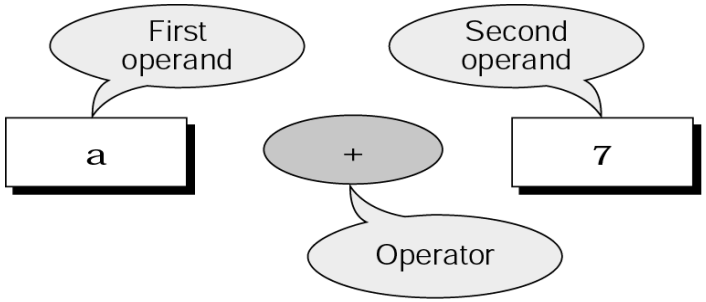
Primary Expressions

- The are examples of three types of primary expressions.
  - Identifier
  - Constant
  - Expression
- Complex expressions enclosed in parentheses are considered primary expressions.
- Sometimes a C++ programmer will use an extra set of parentheses the guarantee the sequence of evaluation whenever it is needed or just for clarity.



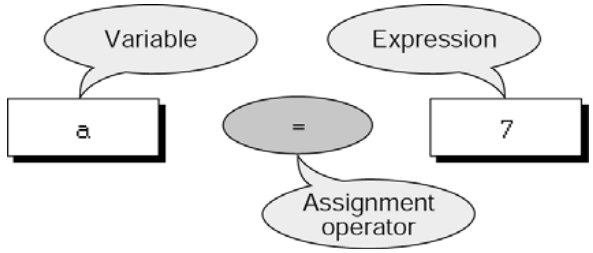
Binary Expressions

- This is an 'operand – operator – operand' combination
- For modulus, both operands must of integer types!

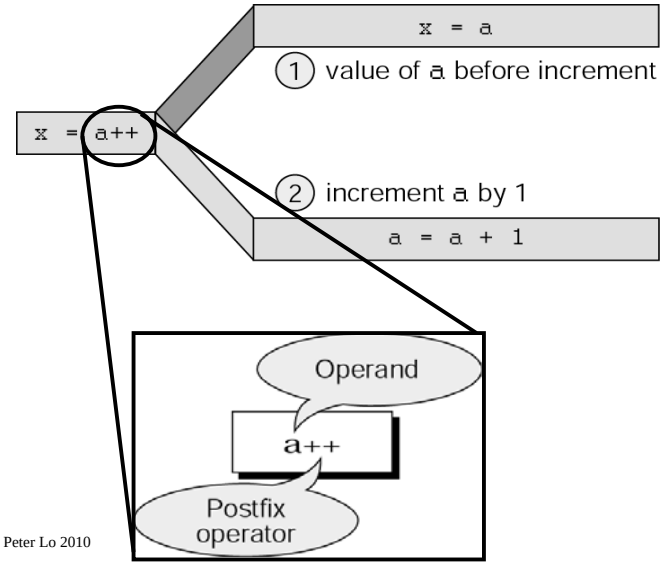


Assignment Expressions

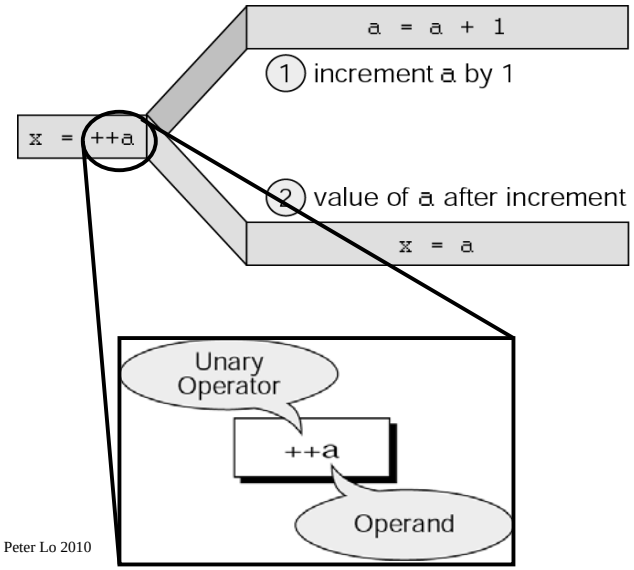
- The assignment expression has a value and a result.
  - The value of the total expression is the value of the expression on the right of the assignment operator (=).
- The result places the expression value in the operator on the left of the assignment operator.
- The left operand in an assignment expression must be a single variable. This is known as an "l-value".



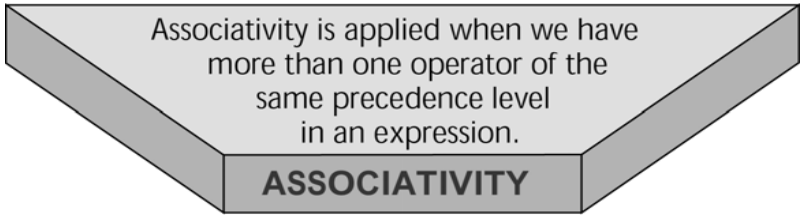
Postfix Expressions



Unary Expressions

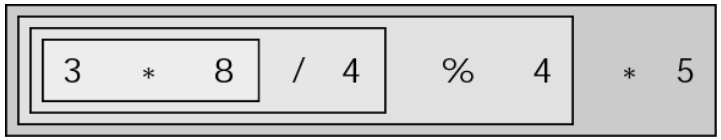


Associativity



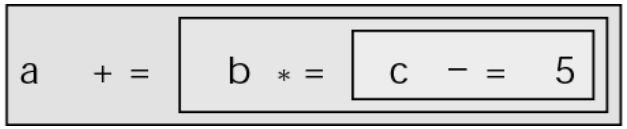
Left Associativity

- Evaluate the expression from the left to the right.This is the most common form.
- The judicious use of parentheses is a good technique to clarify your intent. Parentheses can also be used to override the default evaluation sequence.
- The rules are nearly identical to those you learned in Algebra.



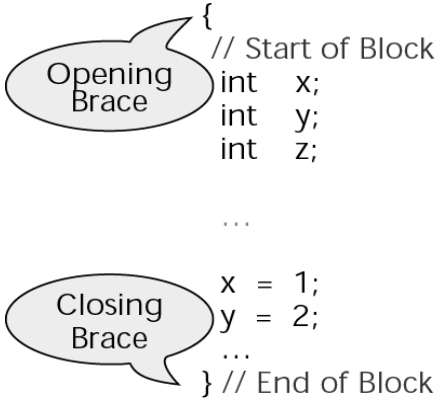
Right Associativity

- The assignment operator is by far the most common occurrence of right associativity.
- The below, while legal, would be considered an example of poor technique. You will learn that just because you can does not mean that you should !



Compound Statement

- An expression statement is terminated with a semicolon. The semicolon is a terminator, and it tells the compiler that the statement is finished

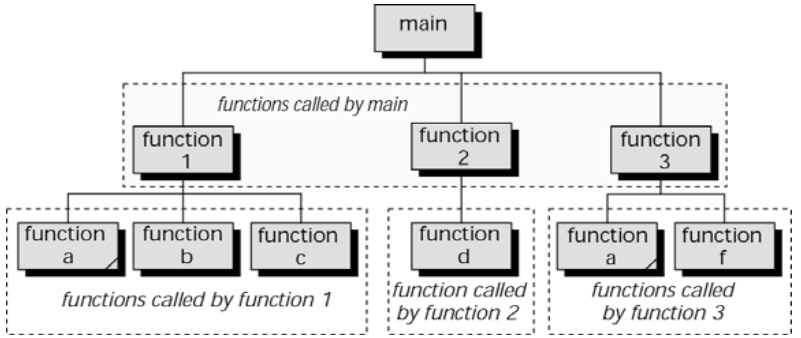


Subroutine

Declare and Calling Subroutine

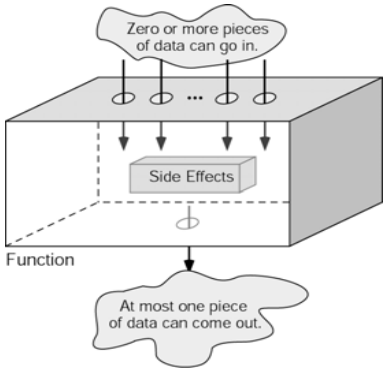
Structure Chart for a C++ Program

- In C++, a program is made of one or more functions, one and only one of which must be named main. The execution of the program always starts with main, but it can call other functions to do some part of the job.



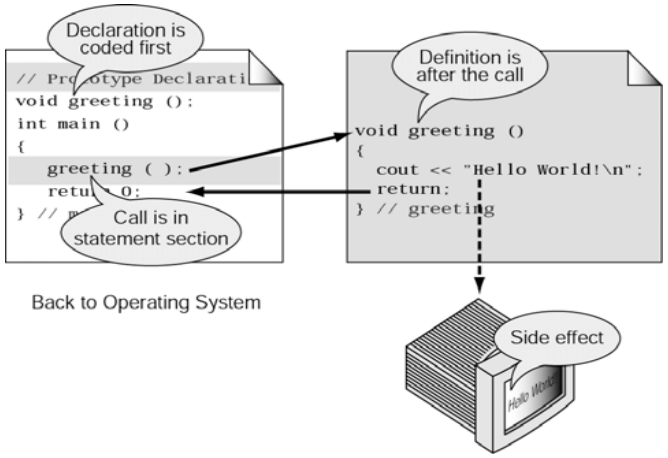
Function Concept

- A function in C++ can have a value, a side effect, or both.
- The side effect occurs before the value is returned.
- The function's value is the value of the expression in the return statement.
- A function can be called for its value, its side effect, or both.

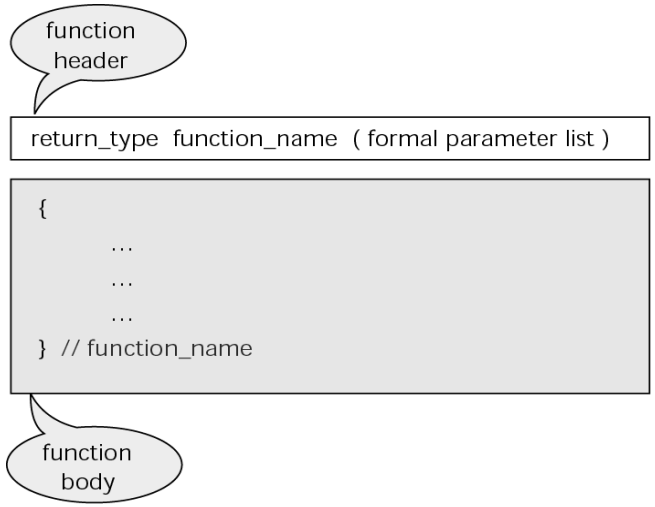


Declaring, Calling, and Defining Functions

- The name of a function is used in three ways: for declaration, in a call, and for definition.

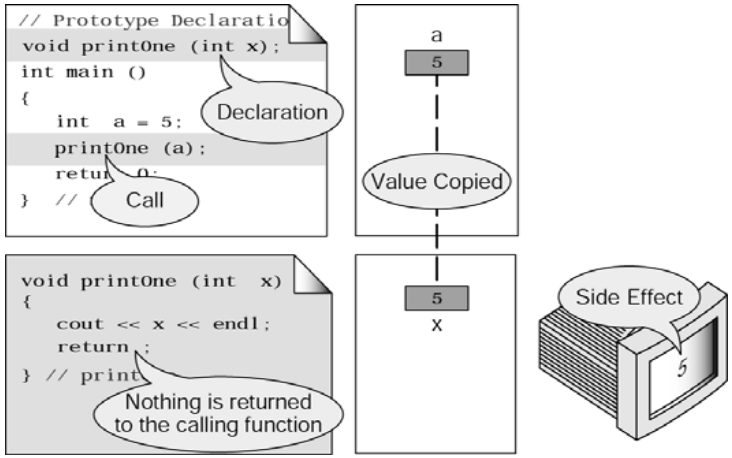


Function Definition



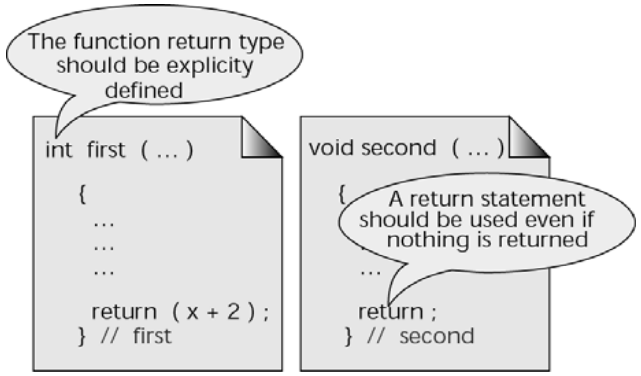
void Function with Parameters

- void functions cannot be used in an expression; they must be a separate statement



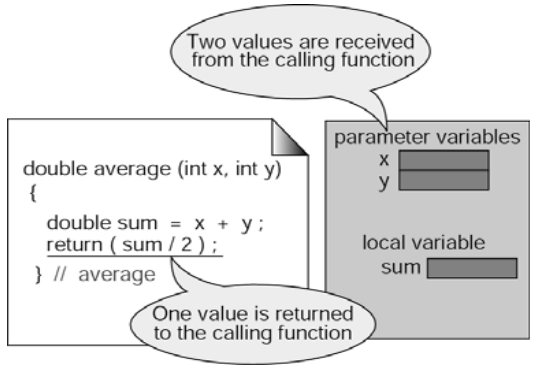
Function Return Statements

- Functions that return a value may be used in an expression or as a separate statement.
- The type of the expression in the return statement must match the return type in the function header



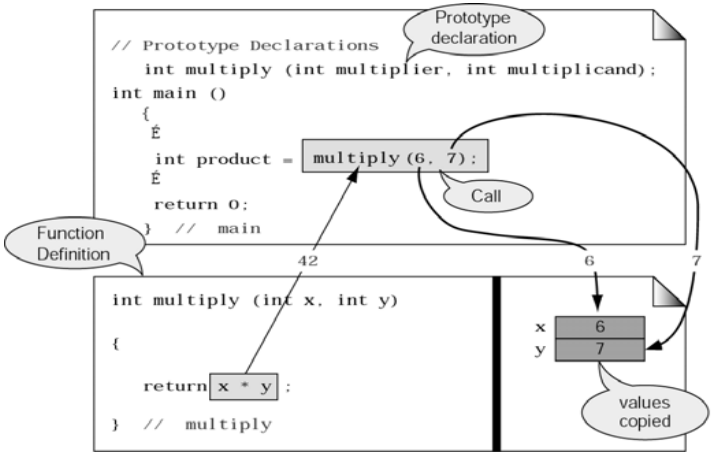
Function Local Variables

- Formal parameters are variables that are declared in the header of the function definition.
- Actual parameters are the expressions in the calling statement.
- The formal and actual parameters must match exactly in type, order, and number.



Parts of a Function Call

- It is the nature of the task to be performed, not the amount of code, that determines if a function should be used.



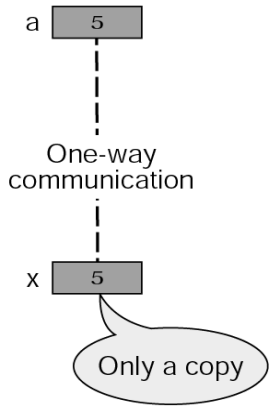
Pass by Value

```
// Prototype Declarations
void fun (int num1);

int main ()
{
   int  a = 5;
   fun (a)
   cout << a << endl;
   return 0;
}  // main
```

prints 5

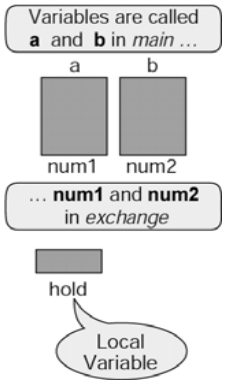
```
void fun (int x)
{
   x = x + 3;
   return;
}  // fun
```



Pass by Reference

```
// Prototype Statements
   void exchange  ( int& num1,
                    int& num2 ) ;
int main ( )
{
   int  a ;
   int  b ;
   ...
   exchange ( a,  b ) ;
   cout << a << " " << b << endl;
   ...
   return 0 ;
}  // main
```

```
void exchange  ( int& num1,
                 int& num2 )
{
     int  hold  = num1 ;
          num1  = num2 ;
          num2  = hold ;
     return ;
} // exchange
```



Library Functions and the Linker