H.O.#2
Fall 2015
Gary Chan

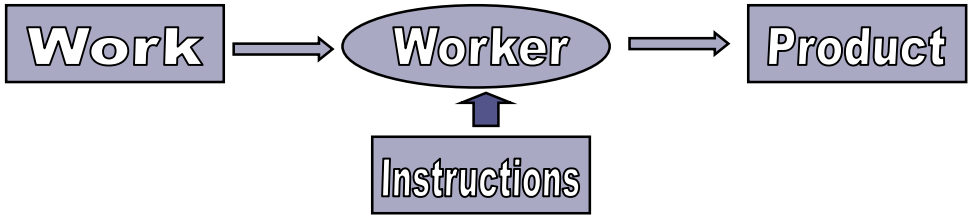
Overview of C++ Programming

Topics

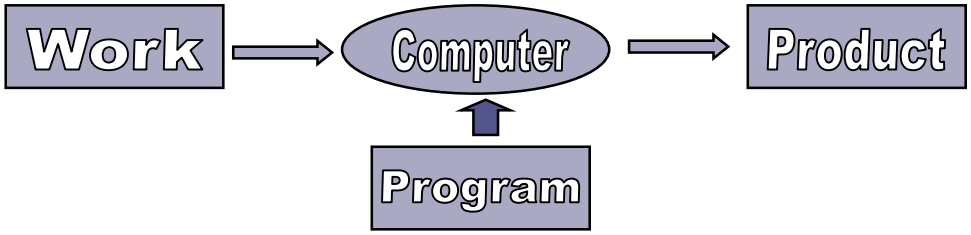
- C++ as a problem-solving tool
- Introduction to C++
  - General syntax
  - Variable declarations and definitions
  - Assignments
  - Arithmetic operators

What is C++?

- C++ is a *programming language*.
- A computer program performs a specific task, and may interact with the user and the computer hardware.
  - Human work model:



  - Computer work model:



What is C++?

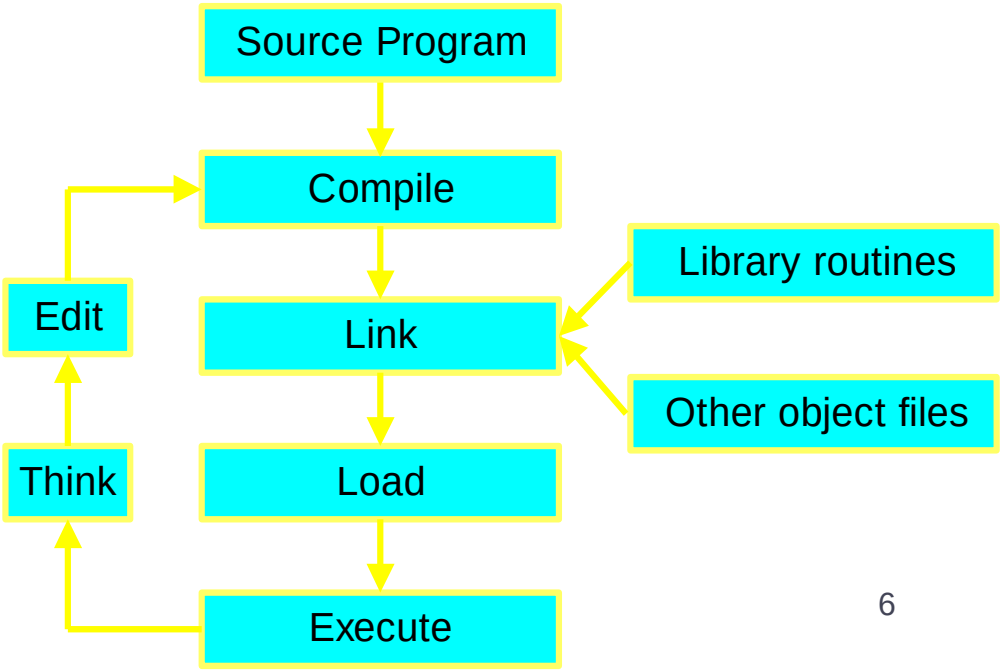
- One of the most popular programming languages:
  - C++ (originally C)
  - Basic
  - Pascal
  - Java
  - Perl
  - Cobol
  - Scheme (Lisp)
  - Smalltalk
- The common versions of C++:
  - Microsoft Visual C++
  - g++ (for Unix machines)

Why C++?

- Bad News:
  - C++ is not so easy to learn
- Good News:
  - Lots of good jobs for programmers
  - Proficient in C++ makes you a good programmer (as it is a very disciplined language)
  - C++ has wide applications in commercial, industrial and government sectors
  - Though C++ is not the easiest language (Java and Python are easier), it is not the hardest either (Scheme, Prolog, and Assembly languages are really difficult!)

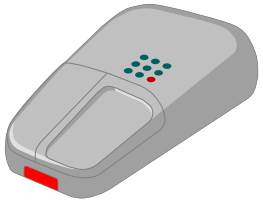
C++ Software Development (More on This Later)

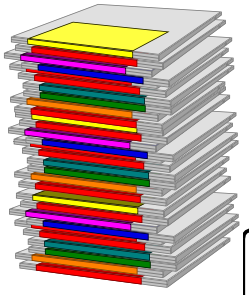
- Major steps
  1. Editing (to write the program)
  2. Compiling (creates .obj file)
  3. Linking with compiled files (creates .exe file)
     - Object files
     - Library modules
  4. Loading and executing
  5. Testing the program

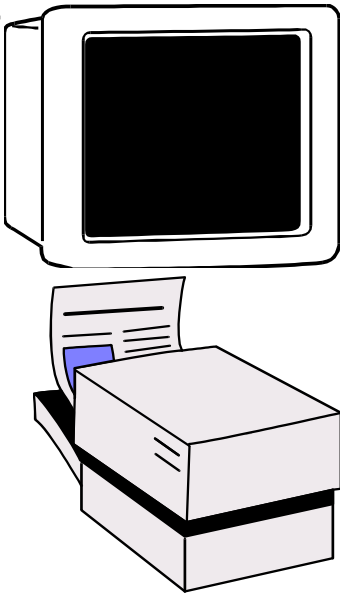


Hardware Overview

The variables and instructions are stored here





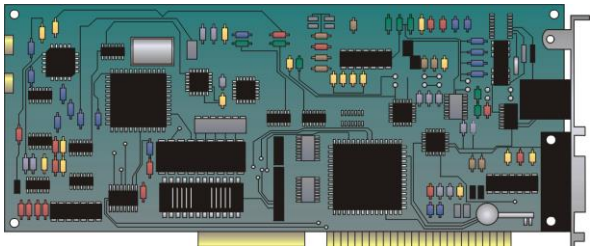


Memory

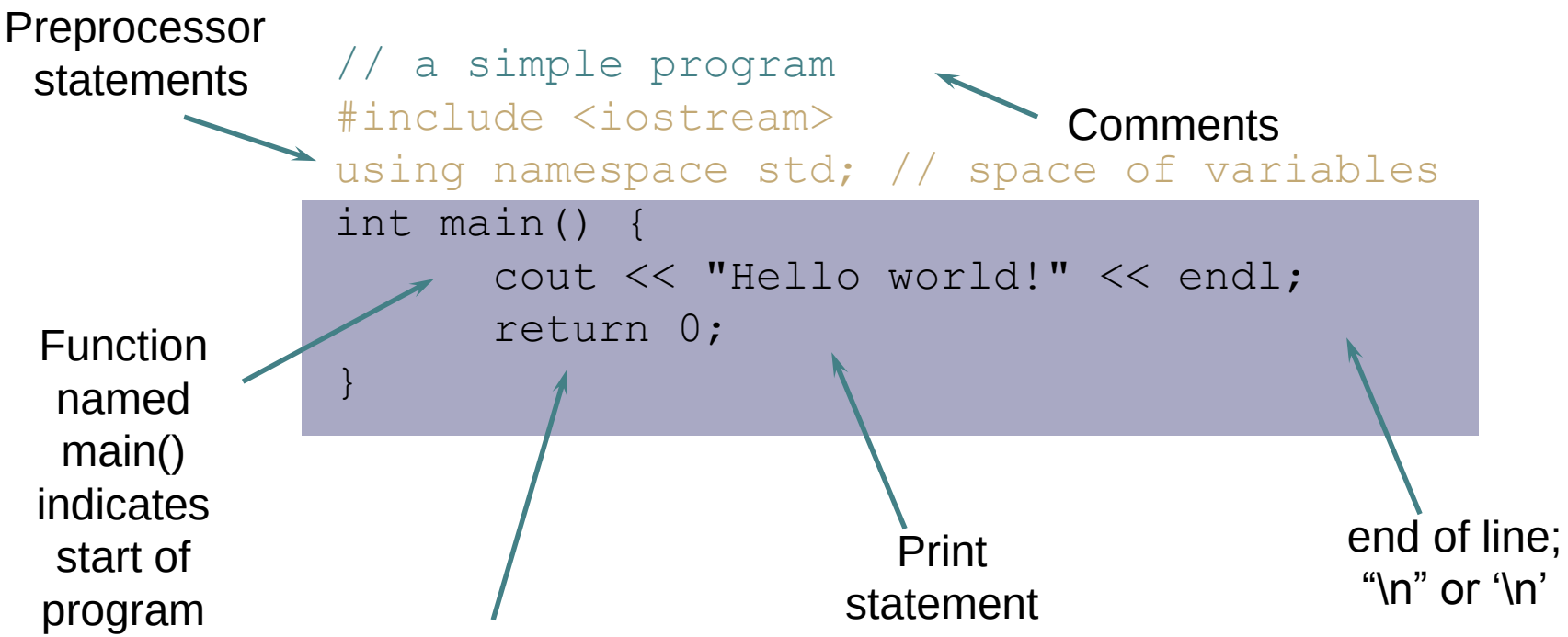
Input Devices

Output Devices

CPU



Our First Program




```
cssu5:~> g++ hello.cpp -o hello
cssu5:~> hello
Hello world!
cssu5:~>
```

using namespace

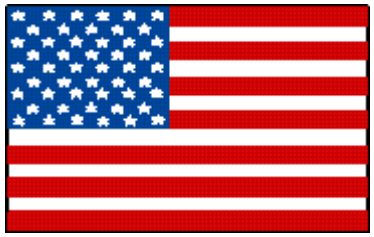
- One of the problems encountered in C is that you run out of names for functions and identifiers
  - It becomes harder to think of new names, e.g., using sort(), sort1(), sort_new(), etc., is not ideal
- In standard C++, each function and identifier lives in a "region" or space, to allow name reuse or prevent naming collision. This is done using `using namespace` keywords.
  - Each set of C++ definitions in a library or program is "wrapped" in a namespace, and if some other definition has an identical name, but is in a different namespace, then there is no collision
  - E.g., `std::cout << "hello\n";  // print to standard out`
- The command `using namespace` is the context under which your program is run. You must be aware of namespace before you write any programs. Without it, if you simply include header file and use some functions or classes from that header, you may get some strange errors.
- All the standard C++ libraries are wrapped in a single namespace, which is `std`.

Programming as Problem Solving

- Define the problem.
  - What is the input & output?
  - What constraints must be satisfied?
  - What information is essential?
- Develop a solution
  - Construct an algorithm (steps that must be done)
  - Implement a program.
- Compile, test, and debug the program.
- Document and maintain the program.

Example

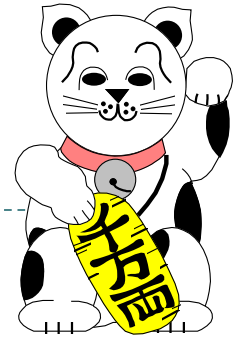
- Problem Statement:
  - Convert US dollars into Hong Kong dollars.
- Problem Analysis:
  - Input: Amount in US$
  - Output: Amount in HK$
  - Apply official currency exchange rates.



Example

- Algorithm
  - Read in amount in US$
  - Calculate the HK$ amount
  - Display the results

Example




```
// converts US$ to HK$
#include <iostream>
using namespace std;
int main(){
    double usdollars;
    double hkdollars;

    // read in amount in US$
    cout <<"Enter US$ amount and press return: ";
    cin >> usdollars;

    // calculate the HK$ amount
    hkdollars = 7.8 * usdollars;

    // display the results
    cout << "US$" << usdollars
         << " = HK$" << hkdollars << endl;
    return 0;
}
```

```
cssu5:~> g++ dollar.cpp -o dollar
cssu5:~> dollar
Enter US$ amount and press return:
2.89
US$2.89 = HK$65.1464
```

Precision Printing

(Printing to 2 decimal points)

```
#include <iostream>
using namespace std;
int main(){
    double x = 123.4567;

    cout.setf(ios::fixed);    // ostream's functions
    cout.precision(2);        // ostream's functions
    cout << x << endl;

    // or printf( "%.2f\n", x );  //include <stdio.h>

    return 0;
}
```

```
cssu5:~> g++ dollar.cpp -o dollar
cssu5:~> a.out
123.45
```

Example

- Program Implementation:
  - Program comments: `//`
  - Library reference: `#include`
  - Function type (for `main`): `int`
  - Function name and (lack of) parameters: `main( )`
  - Statement braces: `{ }`
  - Variable definition: `double`
  - Input/output functions: `cin, cout`

What Makes a Good Program?

- Correctness
  - Meets the problem requirements
  - Produces correct results
- Easy to read and understand
- Easy to modify
- Easy to debug
- Efficient
  - Fast
  - Requires less memory

Basic C++

General form of a C++ program

- C++ is a programming language for manipulating numbers and user-defined objects.

```
//Program description is first
#include directives go next
using namespace std;
int main(){
  constant declarations/definitions go here
  variable declarations/definitions go here

  assignment statements go here
  return 0;
}
```

Declaration vs. Definition

- A variable provides us with named storage that we can write to, retrieve, and manipulate throughout the course of our program
  - Determines the size and layout of its associated memory, the range of values that can be stored within that memory, and the set of operations that can be applied to it
  - We speak of a variable, alternatively, as an object
- The *definition* of an object results in an allocation of storage
  - `int i; double j; char k;`
- The *declaration* of an object makes known the type and name of the object without allocation of memory.
  - An assertion that a definition of the variable exists elsewhere in the program
  - `extern int i; extern foo();`
- A program can contain only *one* definition of an object, but *any* number of object declarations
- The statement `int i` is (unfortunately) sometimes spoken as declaration statement, although defintion statement is more accurate

General form of a C++ program

```
//simple program
#include <iostream>
using namespace std;
int main(){
   // constant declaration statement
   const double Pi = 3.14159;
   // variable declaration statements
   double radius;
   double area;

   // assignment statements
   cout << "Enter circle radius: ";
   cin >> radius;
   area = Pi * radius * radius;
   cout << "Area : " << area << endl;
   return 0;
}
```

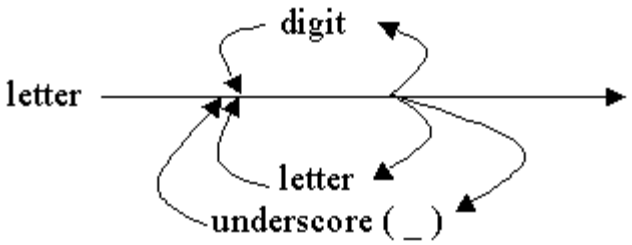
Syntax of the C++ Language

- Reserved words (appear in blue in Visual C++)
  - Reserved words have a special meaning in C++.
  - The list of reserved words:

```
asm, auto, bool, break, case, catch, char,
class, const, continue, default, delete, do,
double, else, enum, extern, float, for,
friend, goto, if, include, inline, int, long,
namespace, new, operator, private, protected,
public, register, return, short, signed,
sizeof, static, struct, switch, template,
this, throw, try, typedef, union, unsigned,
using, virtual, void, volatile, while
```

Syntax of the C++ Language

- Identifiers (appear in black in Visual C++)
  - An identifier is a name for variables, constants, functions, etc.
  - It consists of a letter followed by any sequence of letters, digits or underscores



  - Names are case-sensitive. The following are unique identifiers:

    `Hello, hello, whoami, whoAMI, WhoAmI`
  - Names cannot have special characters in them

    e.g., `X=Y, J-20, #007`, etc. are invalid identifiers.
  - C++ reserved words cannot be used as identifiers.
  - Choose identifiers that are meaningful and easy to remember.

Syntax of the C++ Language

- Comments (appear in green in Visual C++)
  - Comments are explanatory notes; they are not part of the program.
  - Comments are done in two ways:

```
// A double slash starts a single line comment

/* A slash followed by an asterisk marks the
     start of a multiple line comment.
        It ends with an asterisk followed by a slash   */
```

Syntax of the C++ Language

- Compiler Directive: `#include`
  - It refers to a header file of library functions or variables.
  - The compiler reads in the contents of the file before compiling the program.
  - The included file is compiled with the program.
  - There are two forms of `#include`:

```
#include <stdio.h>  // for pre-defined files
#include "my_lib.h" // for user-defined files
```

Libraries

- *#include* loads the code from the standard libraries

```
#include <iostream>   // new I/O library
#include <iostream.h> // old I/O library
#include <stdio.h>    // standard functions
#include <math.h>     // math functions
#include <stdlib.h>   // contains random funct
#include <time.h>     // time function
```

- **Some constant declarations**

```
#define PI 3.14
#define e_CHARGE 1.6e-19
#define EXCH_RATE 7.8
#define PLANK_h 6.626e-34
```

- **`using namespace std;` indicates that the new C++ libraries should be used. If this line is left out, then the old iostream library is loaded:**

```
#include <iostream.h>
```

Constant Definition

- Constants represent permanent values.
- Their values can only be defined/set in the declaration statement:

```
const double pi = 3.14159;
```

- They can make a program more readable and maintainable

Constant definition syntax:

```
const <type> <identifier> = <constant expression>;
```

Examples:

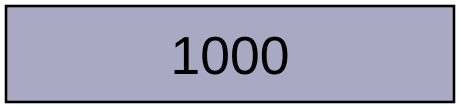
```
const double US2HK = 7.8;
const double HK2Yuan = 0.8;
const double US2Yuan = US2HK* HK2Yuan;
```

Variable Definition

- A variable is best thought of as a container/box for a value:

  Variable definition syntax:

  ```
  <type> <identifier>;
  ```

  Examples:

  ```
  int nickel;
  int penny;
  ```

- A variable must be declared before it can be used.

```
int main(){
        x = 5;      // illegal: x was not declared
}
```

Variable Definitions

- A variable can be initialized in a definition:

```
int x = 3;
```

- Several variables of the same type can be defined in the same declaration statement (though it is sometimes better to put them on separate lines):

```
double total_USD, area;
```

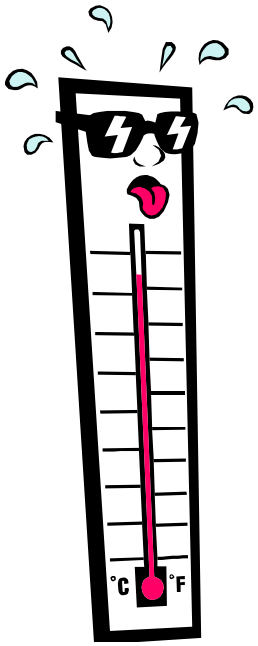
- A variable must have only one type. For example, a variable of the type **`int`** can only hold integer values.

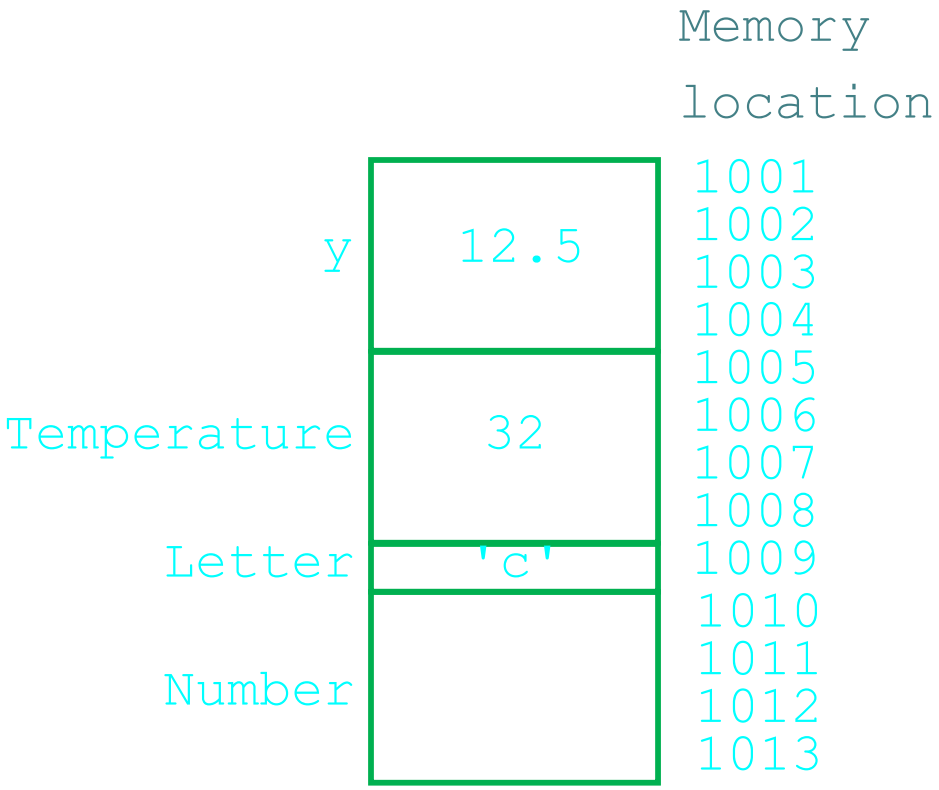
Types of Variable Definitions

- Simple C++ variable types (sizes are often machine-dependent):
  - `int` integer (32-bit integer on PC) (example: 1)
  - `short` 16-bit integer (allows ±32,767)
  - `long` 32-bit integer (allows ±2,147,483,647)
  - `float` 32-bit floating point number (allows about 7 digits of precision: 0.1234567)
  - `double` 64-bit double precision float (allows about 15 digits of precision: 0.12345678901234)
  - `char` 8-bit single character (example: 'y')

Memory Depiction

```
float y = 12.5;
int Temperature = 32;
char Letter = 'c';
int Number;
```





Assignment Statements

Assignment syntax:

```
<identifier> =  <expression>;
```

Examples:

```
int n, m, k;  // definition
n = 5;
m = 6 + (4 * 6);
k = (n * 2) + m;
k = k / 2;
```

Assignment Statements

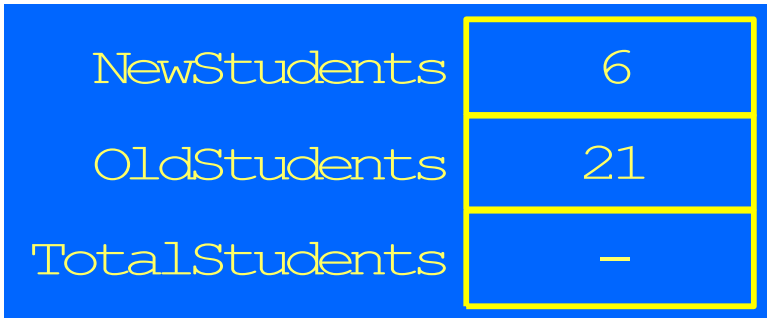
- A variable must be assigned a value before it can be used. Variables are **not** automatically initialized in VC++.

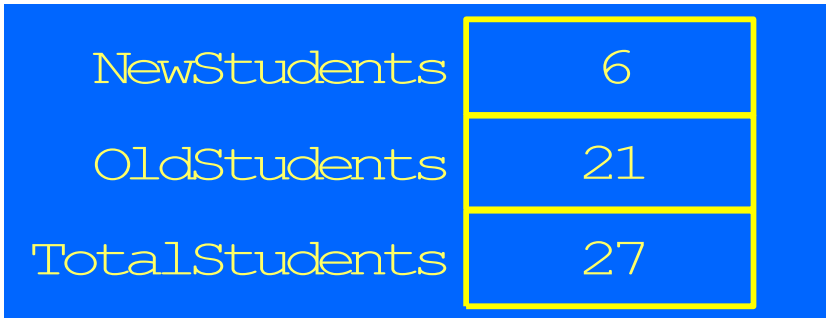
```
int x, y;      // x declared, but not initialized
               // x and y have random values
y = x;  // the random value in x assigned to y
```

- Once a value has been placed in a variable, it stays there until the program changes it.

Assignment Statements

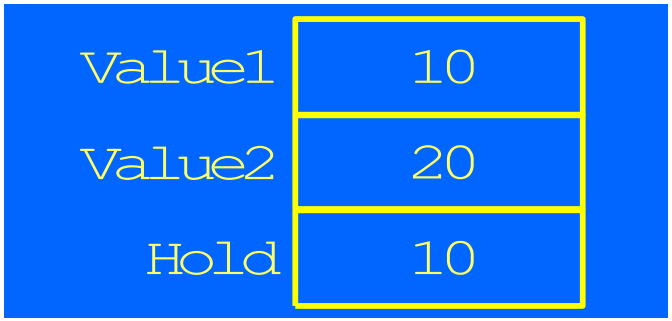
```
int NewStudents = 6;
int OldStudents = 21;
int TotalStudents;

TotalStudents = NewStudents + OldStudents ;
```

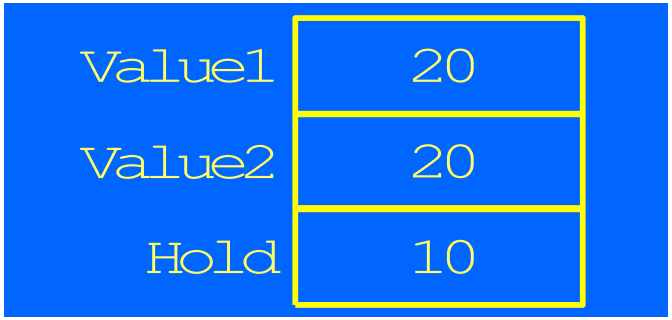


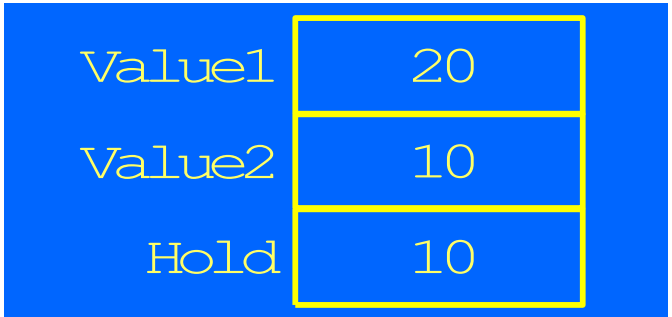


Variable Assignments

```
int Value1 = 10;
int Value2 = 20;

int Hold = Value1;

Value1 = Value2;

Value2 = Hold;
```







Arithmetic Operators

- Common
  - Addition +
  - Subtraction –
  - Multiplication *
  - Division /
  - Mod %
- Note
  - No exponentiation operator

Integer Division

- Integer division produces an integer result
  - It keeps the integral part
- Examples
  - `3 / 2` evaluates to 1
  - `4 / 6` evaluates to 0
  - `10 / 3` evaluates to 3
  - `-10 / 3 = -(10/3)` evaluates to -3

Rules for Division

- C++ treats integers different than doubles.
- 100 is an `int`.
- 100.0 , 100.0000, and 100. are doubles.
- The general rule for division of `int` and `double` types is:
  - `double/double` -> `double` (normal)
  - `double/int` -> `double` (normal)
  - `int/double` -> `double` (normal)
  - `int/int` -> `int` (special case: any decimal places discarded)

Rules for Division

- Example :
  - 220. / 100.0 `double/double -> double` result is 2.2
  - 220. / 100 `double/int -> double` result is 2.2
  - 220 / 100.0 `int/double -> double` result is 2.2
  - 280 / 100 `int/int -> int` result is 2

- Summary: division is normal unless both the numerator and denominator are `int`, then the result is an `int` (the decimal places are discarded).

Forcing a Type Change

- You can change the type of an expression with a cast operation
- Syntax:

```
variable1 = type(variable2);
variable1 = type(expression);
```

- Example:

```
int x=1, y=2;
double result1 = x/y;                    // result1 is 0.0
double result2 = double(x)/y;            // or (double) x / y; result2 is 0.5
double result3 = x/double(y);            // result3 is 0.5
double result4 = double(x)/double(y);    // result4 is 0.5
double result5 = double(x/y);            // result5 is 0.0
```

Remainder

- `a%b` produces the *remainder* (NOT modulo) of the division
  - i.e., `(a/b) * b + a%b` must equal `a` (hence `a%b` is the remainder given by `a - (a/b) * b`)
- Examples

  `5 % 2` evaluates to 1 (=`5-(5/2)*2`)

  `12 % 4` evaluates to 0 (= `12 - (12/4) * 4`)

  `4 % 5` evaluates to 4 (= `4 - (4/5) * 5`)

  `-4 % 5` evaluates to -4 (=`-4 - (-4/5) * 5`)

  `4 % -5` evaluates to 4 (=`4 - (4/(-5))*(-5)`)

  `-4 % -5` evaluates to -4 (= `-4 - (-4/-5)*(-5)`)
- Hence `a%b` is different from mod operator.
  - (a mod b) always returns a non-negative integer, irrespective of the signs of a and b.

Operators and Precedence

- Operator precedence tells how to evaluate expressions
- Standard precedence order
  - `( )` — Evaluated first, if nested innermost done first
  - `* / %` — Evaluated second. If there are several, then evaluate from left-to-right
  - `+ -` — Evaluate third. If there are several, then evaluate from left-to-right
- Examples

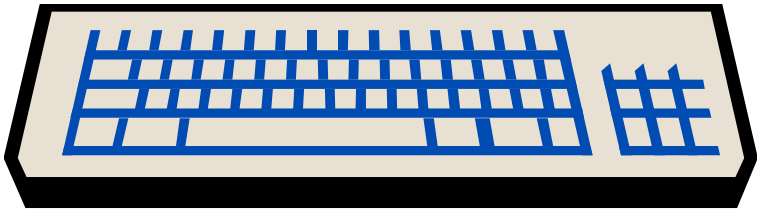
```
1 + 2 * 3 / 4 - 5 = 1 + 6/4 - 5 = -3

2 * 4 / 5 + 3 * 5 % 4 = 8/5 + 15%4 = 1 + 3 = 4

3.0 * 3 / 4 = 9.0/4 = 2.25

(1 + 3) * ((2 + 4 * 6) * 3) / 2 + 2 = 4*( 26*3 )/2 + 2 = 158
```

Standard Input/Output

- `cin` - the standard input stream

Input operator "`>>`"

- extracts data from input "stream" (the keyboard by default)
- skips over white spaces
- extracts only characters of the right form and performs automatic conversion to the type specified



Standard Input/Output

- `cout` - the standard output stream

Output operator "<<"

- inserts data into the output "stream" (the screen by default)
- Example:

```
int id, score;
cout << "Enter student ID and score: ";
cin >> id >> score;
cout << "Student ID: " << id << " score: "
   << score << endl;
```

```
// Convert inches to feet and inches
// Input: inches
// Output: feet and inches
#include <iostream>
using namespace std;
#define PI 3.14

int main() {
   // inches to feet conversion factor
   const int in2feet = 12;

   int inches;              // number of inches
   int feet;                // number of feet

   cout<< "Enter number in inches: ";
   cin >> inches;

   // Convert inches to feet and inches
   feet = inches / in2feet;
   inches = inches % in2feet;
   cout << feet << " feet " << inches << " inches " << endl;
   cout << "Area of circle of radius 2 units is" << PI * 4 << endl;
   return 0;
}
```

Assignment Conversions

- A floating-point expression assigned to an integer object is rounded down
- An integer expression assigned to a floating-point object is converted to a floating-point value
- Example 1:

```
float y = 2.7;
int i = 15;
int j = 10;
i = y;                  // i is now 2
cout << i << endl;
y = j;                  // y is now 10.0
cout << y << endl;
```

Assignment Conversions

- Example 2:

```
int m, n;
double x, y;
m = 3;
n = 2.5;        // 2.5 converted to 2 and assigned to n
x = m/n;        // 3/2=1 converted to 1.0 and assigned to x
n = x+m/2;
   // m/2=1 : integer division
   // x+m/2 : double addition because x is double
   // convert result of m/2 to double (i.e. 1.0)
   // x+m/2=2.0
   // convert result of x+m/2 to int (i.e. 2)
   // because n is int
```

Example: USD to HKD exchange

- Problem Statement
  - Given a collection of nickels (US 5-cent piece) and pennies (US 1-cent piece), find the equivalent number of Hong Kong dollars and 10-cent pieces.
- Problem Analysis
  - Input:
    - nickels (integer): number of US nickels (5 cents)
    - pennies (integer): number of US pennies (1 cent)
    - A US dime is 10 cents
  - Output:
    - dollars (integer): number HK dollar coins to return
    - houji (integer): number HK 10-cent coins to return
- Constraints: None

Example: Initial Algorithm

1. Read in the numbers of nickels and pennies
2. Compute the total value in integral US dollars
3. Exchange the integral USD to HK dollars based on exchange rate
4. Find the number of HK dollar coins and houji coins
5. Display the results

Example: Program Skeleton

```
// File: excoin.cpp
// Determines the number of HK coins to exchange for US coins
#include <iostream>
using namespace std;
int main(){
   int nickel;       // number of nickels
   int penny;        // number of pennies
   int dollar;       // number of HK dollar coins
   int houji;        // number of HK 10-cent coins
   double total_USD; // total value in US$
   double total_HKD; // total value in HK$
   // Read in the number of nickels and pennies
   // Compute the total value in US$
   // Compute the total value in HK$ to exchange
   // Find the numbers of HK dollar and 10-cent coins
   // Display the numbers of HK dollar and 10-cent coins
   return 0;
}
```

Example: Refined Algorithm

1. Read in the number of nickels and pennies
2. Compute the total value in US dollars

   total_USD = (5 * nickel + penny)/100
3. Compute the total in HK dollars to exchange

   total_HKD = total_USD * US2HK
4. Find the number of HK dollar coins and 10-cent coins

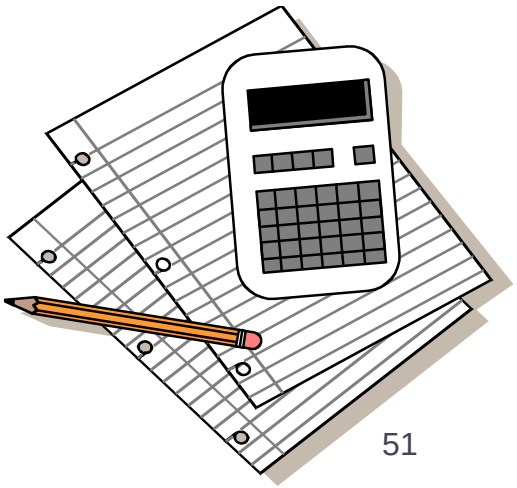
   total_HK_cent = total_HKD * 100

   dollar = total_HK_cent / 100

   houji = (total_HK_cent % 100) / 10
5. Display the number of HK dollar and 10-cent coins

C++ Arithmetic Operators

- The four operators +, −, *, and / work as we expect with the "normal" precedence rules (e.g., 5+2*3 = 11)
- Parenthesis can be inserted to change the order of operations (e.g., (5+2)*3 = 21)
- Be careful of integer division -- any remainder is discarded
- The % (modulo) operator gives the remainder of integer division



```
// File: excoin.cpp
// Determines the number of HK coins to exchange for US coins
#include <iostream>
using namespace std;
int main(){
   const double US2HK = 7.8;        // assume exchange rate is US$1 = HK$7.8
   int nickel;                      // number of nickels
   int penny;                       // number of pennies
   int dollar;                      // number of HK dollar coins
   int houji;                       // number of HK 10-cent coins
   double total_USD;                // total value in US$
   int total_HK_cent;               // total value in HK cents
   double total_HKD;                // total value in HK$

   // Read in the number of nickels and pennies
   cout << "Enter the number of nickels and press return: ";
   cin >> nickel;
   cout << "Enter the number of pennies and press return: ";
   cin >> penny;
```

```
  // Compute the total value in US$
  total_USD = (5 * nickel + penny) / 100.0;

  // Compute the total value in HK$ using the assumed exchange rate
  total_HKD = total_USD * US2HK;

  // Find the value in HK dollars and change
  total_HK_cent = total_HKD * 100;
  dollar = total_HK_cent / 100;
  houji = (total_HK_cent % 100)/10;

  // Display the number of HK dollar and 10-cent coins
  cout << "The change is HK " << dollar << " dollars and "
       << houji << " 10-cent coins." << endl;
  return 0;
}
```

A faster way:

```
dollar = total_HKD; // implicit cast
houji = 10 * (total_HKD - dollar);
```

C++ is a Free-Format Language

- Extra blanks or tabs are ignored

```
x=3;
x       =                3       ;
```

- Blank lines are ignored just like comments
- Code can be indented in any way
- More than one statement can be on one line

```
int x, y;  x=3;  y = 10;       int z = x+y;
```

- A single statement can be continued over several lines

```
int x
  =
                2;
cout << feet << " feet and "
          << inches << " inches" << endl;
```

Good Programming Style

- Place each statement on a line by itself (except long `cout` statements)

```
x = m/n;
```

- Use blank lines to separate sections of code
- Use the same indentation for all statements in the same block of code {…}.

```
int main(){
        int x;
        x = 5;
        return 0;
}
```

Good Programming Style

- Use meaningful identifier names

```
double area, sum, radius;
```

- Document each variable when it is declared

```
double area;            // area of the circle
double distance;        // distance from top to bottom
```

- Document each segment of code

```
// Convert inches to feet and inches
feet = inches / in2feet;
inches = inches % in2feet;
```

- Document the beginning of the program with a header that tells the purpose of the program

```
// Convert inches to feet and inches
```