THE PART OF A C++ PROGRAM

// sample C++ program comment

#include <iostream> preprocessor directive

using namespace std; which namespace to use

int main() beginning of function named main

{ beginning of block for main

    cout << "Hello, there!"; output statement string literal

    return 0; send 0 to operating system

} end of block for main
THE PART OF A C++ PROGRAM

SPECIAL CHARACTERS

<table>
<thead>
<tr>
<th>Character</th>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>//</td>
<td>Double slash</td>
<td>Beginning of a comment</td>
</tr>
<tr>
<td>#</td>
<td>Pound sign</td>
<td>Beginning of preprocessor directive</td>
</tr>
<tr>
<td>&lt; &gt;</td>
<td>Open/close brackets</td>
<td>Enclose filename in #include</td>
</tr>
<tr>
<td>( )</td>
<td>Open/close parentheses</td>
<td>Used when naming a function</td>
</tr>
<tr>
<td>{ }</td>
<td>Open/close brace</td>
<td>Encloses a group of statements</td>
</tr>
<tr>
<td>&quot; &quot;</td>
<td>Open/close quotation marks</td>
<td>Encloses string of characters</td>
</tr>
<tr>
<td>;</td>
<td>Semicolon</td>
<td>End of a programming statement</td>
</tr>
</tbody>
</table>

THE COUT OBJECT

• Displays output on the computer screen
  • You use the stream insertion operator << to send output to cout:

    cout << "Programming is fun!";

• Can be used to send more than one item to cout:

    cout << "Hello " << "there!";

• Or:

    cout << "Hello ";
    cout << "there!";

• This produces one line of output:

    cout << "Programming is ";
    cout << "fun!";
THE COUT OBJECT

THE ENDL MANIPULATOR

• You can use the endl manipulator to start a new line of output. This will produce two lines of output:

```cpp
    cout << "Programming is" << endl;
    cout << "fun!";
```

• You do NOT put quotation marks around endl
• The last character in endl is a lowercase L, not the number 1.

```
endl ------------ This is a lowercase L
```

THE COUT OBJECT

THE \n ESCAPE SEQUENCE

• You can also use the \n escape sequence to start a new line of output. This will produce two lines of output:

```cpp
    cout << "Programming is\n";
    cout << "fun!";
```

Notice that the \n is INSIDE the string.

```
Programming is
fun!
```
THE #INCLUDE DIRECTIVE

- Inserts the contents of another file into the program
- This is a preprocessor directive, not part of C++ language
- `#include` lines not seen by compiler
- Do not place a semicolon at end of `#include` line

VARIABLES AND LITERALS

**VARIABLE**

- a storage location in memory
  - Has a name and a type of data it can hold
  - Must be defined before it can be used:

```
int item;
```

<table>
<thead>
<tr>
<th>Program 2-7</th>
</tr>
</thead>
<tbody>
<tr>
<td>// This program has a variable.</td>
</tr>
<tr>
<td>#include &lt;iostream&gt;</td>
</tr>
<tr>
<td>using namespace std;</td>
</tr>
<tr>
<td>int main()</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>int number;</td>
</tr>
<tr>
<td>number = 5;</td>
</tr>
<tr>
<td>cout &lt;&lt; &quot;The value in number is &quot; &lt;&lt; number &lt;&lt; endl;</td>
</tr>
<tr>
<td>return 0;</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

**Program Output**
The value in number is 5
VARIABLES AND LITERALS

LITERALS

- a value that is written into a program's code.
  - "hello, there" (string literal)
  - 12 (integer literal)

Program 2-9

```cpp
1 /\ This program has literals and a variable.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7   int apples;
8   apples = 20;
9   cout << "Today we sold " << apples << " bushels of apples.\n";
10  return 0;
11 }
```

Program Output

Today we sold 20 bushels of apples.

20 is an integer literal

These are string literals

Program Output

Today we sold 20 bushels of apples.
IDENTIFIERS

DEFINITION

• An identifier is a programmer-defined name for some part of a program: variables, functions, etc.

C++ KEY WORDS

• You cannot use any of the C++ key words as an identifier.
  • These words have reserved meaning.

<table>
<thead>
<tr>
<th>C++ Key Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
</tr>
<tr>
<td>default</td>
</tr>
<tr>
<td>delete</td>
</tr>
<tr>
<td>dynamic_cast</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>extern</td>
</tr>
<tr>
<td>friend</td>
</tr>
<tr>
<td>protected</td>
</tr>
<tr>
<td>true</td>
</tr>
</tbody>
</table>

www.unc.edu.pk
Air University, Islamabad
www.merit bishop.com
IDENTIFIERS

VARIABLE NAMES

- A variable name should represent the purpose of the variable. For example:

  \texttt{itemsOrdered}

- The purpose of this variable is to hold the number of items ordered.

IDENTIFIERS

IDENTIFIER RULES

- The first character of an identifier must be an alphabetic character or an underscore (\_).

- After the first character you may use alphabetic characters, numbers, or underscore characters.

- Upper- and lowercase characters are distinct.
**IDENTIFIERS**

**VALID AND INVALID IDENTIFIERS**

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>VALID?</th>
<th>REASON IF INVALID</th>
</tr>
</thead>
<tbody>
<tr>
<td>totalSales</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>total_Sales</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>total.Sales</td>
<td>No</td>
<td>Cannot contain .</td>
</tr>
<tr>
<td>4thQtrSales</td>
<td>No</td>
<td>Cannot begin with digit</td>
</tr>
<tr>
<td>totalSale$</td>
<td>No</td>
<td>Cannot contain $</td>
</tr>
</tbody>
</table>

**INTEGER DATA TYPES**

- Integer variables can hold whole numbers such as 12, 7, and -99

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Size</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>short</td>
<td>2 bytes</td>
<td>-32,768 to +32,767</td>
</tr>
<tr>
<td>unsigned short</td>
<td>2 bytes</td>
<td>0 to +65,535</td>
</tr>
<tr>
<td>int</td>
<td>4 bytes</td>
<td>-2,147,483,648 to +2,147,483,647</td>
</tr>
<tr>
<td>unsigned int</td>
<td>4 bytes</td>
<td>0 to 4,294,967,295</td>
</tr>
<tr>
<td>long</td>
<td>4 bytes</td>
<td>-2,147,483,648 to +2,147,483,647</td>
</tr>
<tr>
<td>unsigned long</td>
<td>4 bytes</td>
<td>0 to 4,294,967,295</td>
</tr>
</tbody>
</table>
INTEGER DATA TYPES

DEFINING VARIABLES

• Variables of the same type can be defined
  
  • On separate lines:
    
    int length;
    int width;
    unsigned int area;
  
  • On the same line:
    
    int length, width;
    unsigned int area;
  
• Variables of different types must be in different definitions

INTEGER DATA TYPES

INTEGER TYPES IN PROGRAM

Program 2-10

1 // This program has variables of several of the integer types.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     int checking;
8     unsigned int miles;
9     long days;
10
11     checking = -20;
12     miles = 4276;
13     days = 189000;
14     cout << "We have made a long journey of " << miles;
15     cout << " miles.\n";
16     cout << "Our checking account balance is " << checking;
17     cout << "\nAbout " << days << " days ago Columbus \n";
18     cout << "stayed on this spot.\n";
19     return 0;
20 }
INTEGER DATA TYPES

INTEGER LITERALS

- An integer literal is an integer value that is typed into a program's code. For example:

  ```
  itemsOrdered = 15;
  ```

- In this code, 15 is an integer literal.

---

### INTEGER LITERALS IN PROGRAM

**Program 2-10**

```cpp
1 // This program has variables of several of the integer types.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7   int checking;
8   unsigned int miles;
9   long days;
10  
11   checking = -20;
12   miles = 4276;
13   days = 189000;
14   cout << "We have made a long journey of " << miles;
15   cout << " miles.\n";
16   cout << "Our checking account balance is " << checking;
17   cout << " days ago Columbus.\n";
18   cout << "I\'d like to step on this spot.\n";
19   return 0;
20 }
```
INTEGER DATA TYPES

INTEGER LITERALS

- Integer literals are stored in memory as `int`s by default

- To store an integer constant in a long memory location, put ‘L’ at the end of the number: `1234L`

- Constants that begin with ‘0’ (zero) are base 8: `075`

- Constants that begin with ‘0x’ are base 16: `0x75A`

THE CHAR DATA TYPE

CHARACTER LITERALS

- Used to hold characters or very small integer values
- Usually 1 byte of memory
- Numeric value of character from the character set is stored in memory:

CODE:
```c
char letter;
letter = 'C';
```

MEMORY: `letter` 67

- Character literals must be enclosed in single quote marks. Example: `'A'`
THE CHAR DATA TYPE
CHARACTER LITERALS IN PROGRAM

Program 2.13

1 // This program uses character literals.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     char letter;
8     letter = 'A';
9     cout << letter << endl;
10    letter = 'B';
11    cout << letter << endl;
12    return 0;
13 }

Program Output
A
B

THE CHAR DATA TYPE
CHARACTER STRING

• A series of characters in consecutive memory locations:
  "Hello"

• Stored with the null terminator, \0, at the end:

• Comprised of the characters between the " "

   H e l l o \0
THE C++ STRING CLASS

- Special data type supports working with strings
  
  ```
  #include <string>
  ```

- Can define string variables in programs:
  ```
  string firstName, lastName;
  ```

- Can receive values with assignment operator:
  ```
  firstName = "George";
  lastName = "Washington";
  ```

- Can be displayed via cout
  ```
  cout << firstName << " " << lastName;
  ```

THE C++ STRING CLASS IN PROGRAM

Program 2-15

```cpp
1 // This program demonstrates the string class.
2 #include <iostream>
3 #include <string> // Required for the string class.
4 using namespace std;
5
6 int main()
7 {
8    string movieTitle;
9
10    movieTitle = "Wheels of Fury";
11    cout << "My favorite movie is " << movieTitle << endl;
12    return 0;
13 }
```

Program Output

My favorite movie is Wheels of Fury
FLOATING-POINT DATA TYPES

• The floating-point data types are:
  
  ```c
  float
double
long double
  ```

• They can hold real numbers such as:
  
  12.45  -3.8

• Stored in a form similar to scientific notation

• All floating-point numbers are signed

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Key Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single precision</td>
<td>float</td>
<td>4 bytes, Numbers between ±3.4E-38 and ±3.4E38</td>
</tr>
<tr>
<td>Double precision</td>
<td>double</td>
<td>8 bytes, Numbers between ±1.7E-308 and ±1.7E308</td>
</tr>
<tr>
<td>Long double precision</td>
<td>long double</td>
<td>8 bytes, Numbers between ±1.7E-308 and ±1.7E308</td>
</tr>
</tbody>
</table>

FLOATING-POINT LITERALS

• Can be represented in

  • Fixed point (decimal) notation:
    
    31.4159  0.0000625

  • E notation:
    
    3.14159E1  6.25e-5

• Are double by default

• Can be forced to be float (3.14159f) or long double (0.0000625L)
FLOATING-POINT DATA TYPES

IN PROGRAM

Program 2.16

```cpp
1 // This program uses floating point data types.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7    float distance;
8    double mass;
9
10   distance = 1.495979E11;
11   mass = 1.989E30;
12   cout << "The Sun is " << distance << " meters away.\n";
13   cout << "The Sun\'s mass is " << mass << " kilograms.\n";
14   return 0;
15 }
```

Program Output

The Sun is 1.49598e+011 meters away.
The Sun’s mass is 1.989e+030 kilograms.

THE BOOL DATA TYPE

- Represents values that are true or false
- bool variables are stored as small integers
- false is represented by 0, true by 1:
  ```cpp
  bool allDone = true;
  bool finished = false;
  ```
THE BOOL DATA TYPE

IN PROGRAM

Program 2-17

1 // This program demonstrates boolean variables.
2 #include <iostream>
3 using namespace std;
4 
5 int main()
6 {
7     bool boolValue;
8     boolValue = true;
9     cout << boolValue << endl;
10    boolValue = false;
11    cout << boolValue << endl;
12    return 0;
13 }

Program Output
1
0

DETERMINING THE SIZE OF A DATA TYPE

• The sizeof operator gives the size of any data type or variable:

double amount;
cout << "A double is stored in 
    " << sizeof(double) << "bytes\n";
cout << "Variable amount is stored in 
    " << sizeof(amount) << "bytes\n";
VARIABLE ASSIGNMENTS AND INITIALIZATION

ASSIGNMENT

- An assignment statement uses the = operator to store a value in a variable.
  
  ```
  item = 12;
  ```

- This statement assigns the value 12 to the item variable.

- The variable receiving the value must appear on the left side of the = operator.

- This will NOT work:
  
  ```
  // ERROR!
  12 = item;
  ```

VARIABLE ASSIGNMENTS AND INITIALIZATION

VARIABLE INITIALIZATION

- To initialize a variable means to assign it a value when it is defined:
  
  ```
  int length = 12;
  ```

- Can initialize some or all variables:
  
  ```
  int length = 12, width = 5, area;
  ```

Program 2-19

```cpp
// This program shows variable initialization.
#include <iostream>
using namespace std;

int main()
{
    int month = 2, days = 28;
    cout << "Month " << month << " has " << days << " days.\n";
    return 0;
}
```

Program Output

Month 2 has 28 days.
SCOPE

- The scope of a variable: the part of the program in which the variable can be accessed
- A variable cannot be used before it is defined

Program 2-20

```cpp
1  // This program can't find its variable.
2  #include <iostream>
3  using namespace std;
4  
5  int main()
6  {
7      cost << value; // ERROR! value not defined yet!
8  
9      int value = 100;
10     return 0;
11  }
```

ARITHMETIC OPERATORS

- Used for performing numeric calculations

- C++ has unary, binary, and ternary operators:
  - unary (1 operand) -5
  - binary (2 operands) 13 - 7
  - ternary (3 operands) `exp1 ? exp2 : exp3`
ARITHMETIC OPERATORS

 Binary Arithmetic Operators

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>OPERATION</th>
<th>EXAMPLE</th>
<th>VALUE OF ans</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
<td>ans = 7 + 3;</td>
<td>10</td>
</tr>
<tr>
<td>−</td>
<td>subtraction</td>
<td>ans = 7 − 3;</td>
<td>4</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
<td>ans = 7 * 3;</td>
<td>21</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
<td>ans = 7 / 3;</td>
<td>2</td>
</tr>
<tr>
<td>%</td>
<td>modulus</td>
<td>ans = 7 % 3;</td>
<td>1</td>
</tr>
</tbody>
</table>

ARITHMETIC OPERATORS

In Program

Program 2.21

```cpp
1 // This program calculates hourly wages, including overtime.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7 double regularWages, /* To hold regular wages*/
8   basePayRate = 18.25, /* Base pay rate*/
9   regularHours = 40.0, /* Hours worked less overtime*/
10  overtimeWages, /* To hold overtime wages*/
11   overtimeRate = 27.78, /* Overtime pay rate*/
12   overtimeHours = 10, /* Overtime hours worked*/
13   totalWages; /* To hold total wages*/
14
15  // Calculate the regular wages.
16  regularWages = basePayRate * regularHours;
17
18  // Calculate the overtime wages.
19  overtimeWages = overtimeRate * overtimeHours;
20
21  // Calculate the total wages.
22  totalWages = regularWages + overtimeWages;
23
24  // Display the total wages.
25  cout << "Wages for this week are $" << totalWages << endl;
26  return 0;
27 }
```

Program Output

Wages for this week are $1007.8
ARITHMETIC OPERATORS
A CLOSER LOOK AT THE / OPERATOR

- / (division) operator performs integer division if both operands are integers
  
  ```
  cout << 13 / 5; // displays 2
  cout << 91 / 7; // displays 13
  ```

- If either operand is floating point, the result is floating point
  
  ```
  cout << 13 / 5.0; // displays 2.6
  cout << 91.0 / 7; // displays 13.0
  ```

ARITHMETIC OPERATORS
A CLOSER LOOK AT THE % OPERATOR

- % (modulus) operator computes the remainder resulting from integer division
  
  ```
  cout << 13 % 5; // displays 3
  ```

- % requires integers for both operands
  
  ```
  cout << 13 % 5.0; // error
  ```
COMMENTS

- Used to document parts of the program

- Intended for persons reading the source code of the program:
  - Indicate the purpose of the program
  - Describe the use of variables
  - Explain complex sections of code

- Are ignored by the compiler

COMMENTS

SINGLE-LINE COMMENTS

- Begin with `//` through to the end of line:

  ```cpp
  int length = 12; // length in inches
  int width = 15;  // width in inches
  int area;        // calculated area

  // calculate rectangle area
  area = length * width;
  ```
COMMENTS
MULTIPLE - LINE COMMENTS

- Begin with /*, end with */

- Can span multiple lines:
  
  ```
  /* this is a multi-line
   comment
  */
  ```

- Can begin and end on the same line:
  ```
  int area;   /* calculated area */
  ```

NAMED CONSTANTS

- **Named constant (constant variable):**
  - variable whose content cannot be changed during program execution

- Used for representing constant values with descriptive names:
  ```
  const double TAX_RATE = 0.0675;
  const int NUM_STATES = 50;
  ```

- Often named in uppercase letters
NAMED CONSTANTS
IN PROGRAM

Program 2.28

1 // This program calculates the circumference of a circle.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     // Constants
8     const double PI = 3.14159;
9     const double DIAMETER = 10.0;
10    // Variable to hold the circumference
11    double circumference;
12
13    // Calculate the circumference.
14    circumference = PI * DIAMETER;
15
16    // Display the circumference.
17    cout << "The circumference is: " << circumference << endl;
18    return 0;
19 }

Program Output
The circumference is: 31.4159

PROGRAMMING STYLE

• The visual organization of the source code

• Includes the use of spaces, tabs, and blank lines

• Does not affect the syntax of the program

• Affects the readability of the source code

• Common elements to improve readability:
  • Braces {} aligned vertically
  • Indentation of statements within a set of braces
  • Blank lines between declaration and other statements
  • Long statements wrapped over multiple lines with aligned operators
STANDARD AND PRESTANDARD C++

- Older-style C++ programs:
  - Use .h at end of header files:
  - #include <iostream.h>
  - Use #define preprocessor directive instead of const definitions
  - Do not use using namespace convention
  - May not compile with a standard C++ compiler

Program 2-31

```cpp
1 // This program calculates the circumference of a circle.
2 #include <iostream>
3 using namespace std;
4
5 #define PI 3.14159
6 #define DIAMETER 10.0
7
8 int main()
9 {
10    // Variable to hold the circumference
11    double circumference;
12
13    // Calculate the circumference.
14    circumference = PI * DIAMETER;
15
16    // Display the circumference.
17    cout << "The circumference is: " << circumference << endl;
18    return 0;
19 }
```

Program Output

The circumference is: 31.4159