Chapter 2

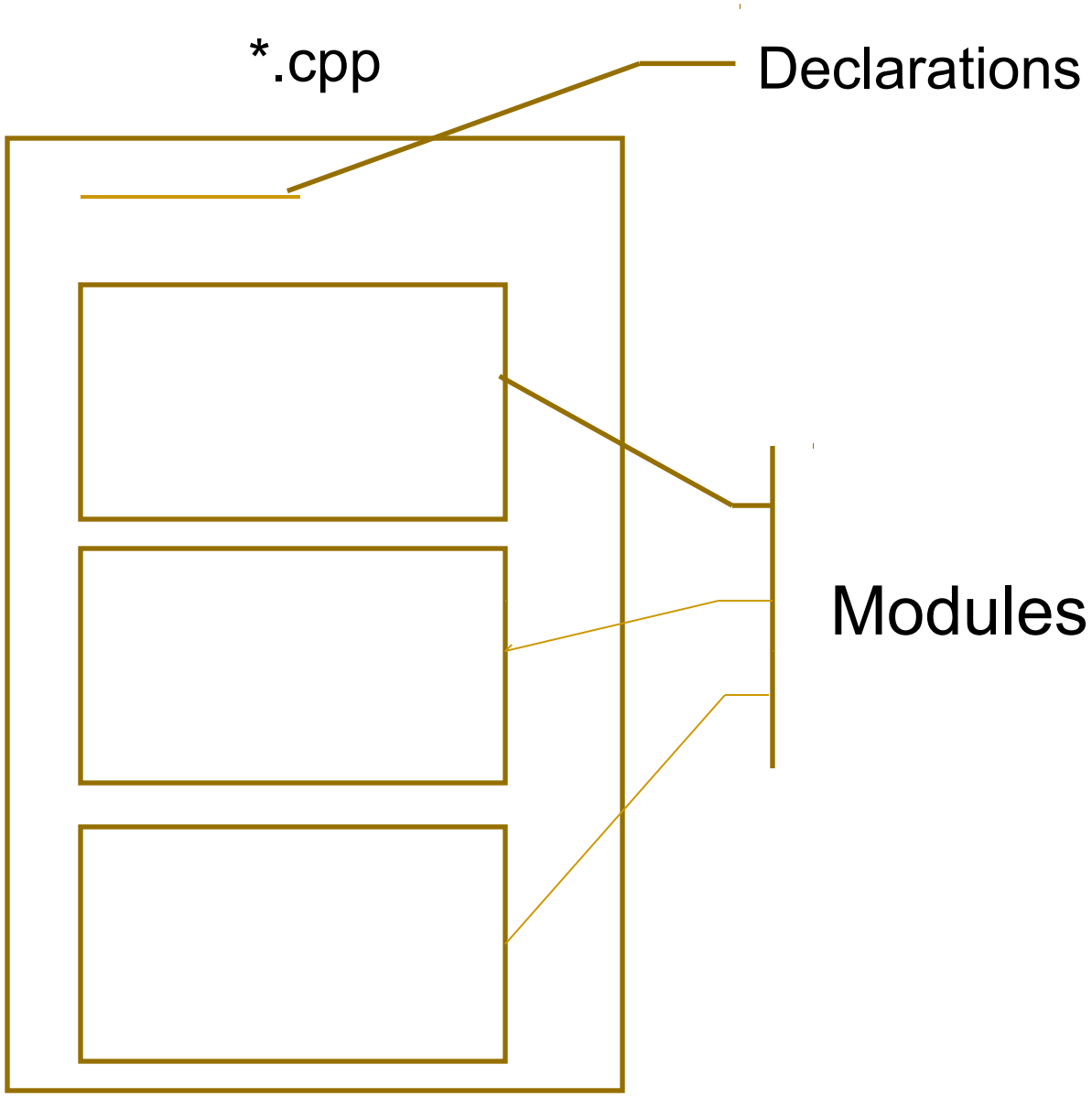
BASIC ELEMENTS IN C++

Chapter 2

- Program structures
- Data types and operators
- Variables, Constants and declaration statements
- Integer quantifiers
- Data Type Conversion

Modular programs

- A large program should be organized as several interrelated segments: The segments are called *modules*.
- A program which consists of such modules is called a modular program.
- In C++, modules can be *classes* or *functions*.
- *A function* is a program segment that transforms the data it receives into a finished result.


*.cpp
Declarations
Modules

Function

- **Each function must have a name.**
- **Names or *identifiers* in C++ can made up of any combination of letters, digits, or underscores selected according to the following rules:**

  - Identifiers must begin within an uppercase or lowercase ASCII letter or an underscore (_).
  - You may use digits in an identifier, but not as the first character.
  - Some special names, i.e. reserved words, cannot be used as a name for a function or variable.

- **Example:**

DegToRad	intersect	addNums
FindMax1	_density	slope

Function declaration

- **Including a function header and a function body**
- **The function header contains three pieces of information:**

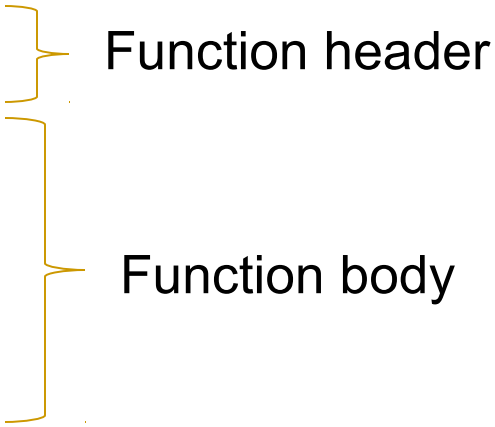
  1. What type of data, if any, is returned from the function.

  2.The name of the function

  3. What type of data, if any, is sent into the function.
- **The function body consists of many statements enclosed in parentheses { }**

Example

```
int  add( int a, int b)
{
    int c;
    c = a + b;
    return c;
}
```



The *main()* function

- **The *main()* function is a special function that runs automatically when a program first executes.**

- **All C++ programs must include one *main()* function. All other functions in a C++ program are executed from the *main()*.**

```
int main()
{
    program statements in here
   return 0;
}
```

The line *return 0;* is included at the end of every *main* function. C++ keyword *return* is one of several means we will use to *exit a function*. When the *return* statement is used at the end of *main()*, the value *0* indicates that the program has terminates successfully.

The cout object

- **The *cout* object is an output object that sends data given to it to the standard output display device.**

- **To send a message to the cout object, you use the following pattern:**

```
cout << "text";
```

- **The *insertion operator*, <<, is used for sending text to an output device.**

- **The text portion of *cout* statement is called a *text string*.**

A simple program

Example 2.1.1

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


header file

- A *header file* is a file with an extension of *.h* that is included as part of a program. It notifies the compiler that a program uses run-time libraries.
- All statements in C++ must end with a semicolon.

The iostream classes

- **The *iostream* classes are used for giving C++ programs input capabilities and output capabilities.**

- **The header file for the iostream class is *iostream.h*.**

- **The *#include* statement is one of the several *preprocessor directives* that are used with C++.**

Example: To include the *iostream.h* file you use the following preprocessor directives:

```
#include <iostream>
```

Preprocessor directives

- **The *preprocessor* is a program that runs before the compiler.**
- **When the preprocessor encounters an *#include* statement, it places the entire contents of the designated file into the current file.**
- **Preprocessor directives and *include* statements allow the current file to use any of the *classes, functions, variables*, and *other code* contained within the included file.**

i/o manipulator

- **An *i/o manipulator* is a special function that can be used with an i/o statement.**
- **The *endl* i/o manipulator is part of *iostream* classes and represents a new line character.**

- **Example:**

```
cout << “Program type: console application” << endl;
cout << “Create with: Visual C++ “<< endl;
cout << “Programmer: Don Gesselin” << endl;
```

Comments

- ***Comments*** **are lines that you place in your code to contain various type of remarks.**
- **C++ line comments are created by adding two slashes (//) before the text you want to use as a comment.**

- **Block comments span multiple lines. Such comments begin with /* and end with the symbols */.**

Example:

```
int main()
{
    /*
     This line is part of the block comment.
     This line is also part of the block
      comment.
    */
    cout << “Line comment 1 “;
    cout << “Line comment 2 “;
    // This line comment takes up an entire line.
   return 0;
}
```

DATA TYPES AND OPERATORS_

Data Types

- **A *data type* is the specific category of information that a variable contains.**
- **There are three basic data types used in C++: integers, floating point numbers and characters.**

Integer data type

- **An integer is a positive or negative number with no decimal places.**
- **Examples:**

  - 259 -13 0 200

Floating Point Numbers

- **A *floating point* number contains decimal places or is written using exponential notations.**

  -6.16 -4.4 2.7541 10.5

- ***Exponential notation*, or scientific notation is a way of writing a very large numbers or numbers with many decimal places using a shortened format.**

  2.0e11 means $2*10^{11}$

The Character Data Type

- **To store text, you use the character data type. To store one character in a variable, you use the *char* keyword and place the character in single quotation marks.**

- **Example:**

  char cLetter = 'A';

- **Escape Sequence**

  The combination of a *backlash* (\) and a special character is called an escape sequence.

- **Example:**

  \n move to the next line

  \t move to the next tab

Arithmetic Operators

- **Arithmetic operators are used to perform mathematical calculations, such as addition, subtraction, multiplication, and division.**

Operator	Description
+	Add two operands
-	Subtracts one operand from another operand
*	Multiplies one operand by another operand
/	Divides one operand by another operand
%	Divides two operands and returns the remainder

- **A simple arithmetic expression consists of an arithmetic operator connecting two operands in the form:**

  operand operator operand

Examples:

3 + 7

18 – 3

12.62 + 9.8

12.6/2.0

Example 2.2.1

```
#include <iostream>
int main()
{
  cout << "15.0 plus 2.0 equals "       << (15.0 + 2.0) << '\n'
       << "15.0 minus 2.0 equals "      << (15.0 - 2.0) << '\n'
       << "15.0 times 2.0 equals "      << (15.0 * 2.0) << '\n'
       << "15.0 divided by 2.0 equals " << (15.0 / 2.0) << '\n';
  return 0;
}
```

Integer Division

- **The division of two integers yields integer result. Thus the value of 15/2 is 7.**

- **Modulus % operator produces the remainder of an integer division.**

- **Example:**

  9%4 is 1

  17%3 is 2

  14%2 is 0

Operator Precedence and Associativity

- **Expressions containing multiple operators are evaluated by the priority, or *precedence*, of the operators.**

Operator	Associativity
unary -	Right to left
* / %	Left to right
+ -	Left to right

- **Example:**

8 + 5*7%2*4

↓ ↓↓ ↓

4 1 2 3

VARIABLES

- **One of the most important aspects of programming is storing and manipulating the values stored in *variables*.**
- **Variable names are also selected according to the rules of identifiers:**
  - Identifiers must begin with an uppercase or lowercase ASCII letter or an underscore (_).
  - You may use digits in an identifier, but not as the first character. You are not allowed to use any other characters.
  - Reserved words cannot be used for variable names.

Identifiers

- **Example: Some valid identifiers**

  my_variable

  Temperature

  x1

  x2

  _my_variable

- **Some invalid identifiers are as follows:**

  %x1 %my_var @x2

Declaration Statements

- **In C++ you can declare the data types of variables using the syntax:**

  type name;

  The *type* portion refers to the data type of the variable.

- **The data type determines the type of information that can be stored in the variable.**

- **Example:**

  ```
  int sum;
  long datenem;
  double secnum;
  ```

Rules of variable declaration

- **Rules:**

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

  2. Declaration statements can also be used to store an initial value into declared variables.

  Example:

  ```
  int num = 15;
  float grade1 = 87.0;
  ```

- **Note: Declaration statement gives information to the compiler**
  - rather than a step in the algorithm.

Example 2.2.1

```
#include <iostream.h>
int main()
{
  float grade1 = 85.5;
  float grade2 = 97.0;
  float total, average;

  total = grade1 + grade2;
  average = total/2.0;  // divide the total by 2.0
  cout << "The average grade is " << average << endl;
  return 0;
}
```

The output of the above program:

```
The average grade is 91.25
```

Assignment statement

- **Let notice the two statements in the above program:**

  total = grade1 + grade2;

  average = total/2.0;

- **Each of these statement is called an *assignment statement* because it tells the computer to assign (store) a value into a variable.**
- **Assignment statements always have an equal (=) sign and one variable name on the left of this sign.**
- **The value on the right of the equal sign is assigned to the variable on the left of the equal sign.**

Display a Variable's Address

- **Every variable has three major items associated with it:**
  - its data type
  - value
  - the address of the variable.

- **To see the address of a variable, we can use *address operator*, &, which means "the address of ".**

- **For example, *&num* means the address of *num*.**

Example 2.2.2

```
#include <iostream.h>
int main()
{
  int num;
  num = 22;
  cout << "The value stored in num is " << num << endl;
  cout << "The address of num = " << &num << endl;
  return 0;
}
```

The output of the above program:

The value stored in num is 22
The address of num = 0x8f5afff4

The display of addresses is in hexadecimal notation.

Constant and the const qualifier

- A constant represents a value provided in a constant definition and cannot change the value.
- To define a constant in a program, we use *const* declaration qualifier.
- Example:

```
const float PI = 3.1416;
const double SALESTAX = 0.05;
const int MAXNUM = 100;
```

- Once defined, a constant can be used in any C++ statement in place of the value it represents.

INTEGER QUANTIFIERS

- **C++ provides *long integer*, *short integer*, and *unsigned integer* data types.**
- **These three additional integer data types are obtained by adding the quantifier *long, short* or *unsigned* to the integer declaration statements.**

- **Example:**

```
long int days;
unsigned int num_of_days;
```

Unsigned integers

- **The reserved words *unsigned int* are used to specify an integer that can only store nonnegative numbers.**

Data type	Storage	Number Range
short int	2 bytes	-32768 to 32767
unsigned int	2 bytes	0 to 65535

Data Type Conversions

- **An expression that contains only integer operands is called an *integer expression*, and the result of the expression is an integer value.**

- **An expression that contains only floating point operands (single and double precision) is called a *floating point expression*, and the result of such an expression is a floating point value.**

- **An expression containing both integer and floating point operands is called a *mixed mode expression*.**
- **Example:**

```
int a;
float x = 2.5;
a = x + 6;          // x + 6 is a mixed mode expression
```

Data Type Conversion Rules

The general rules for converting integer and floating point operands in mixed mode expressions were as follows:

1. **If both operands are either character or integer operands:**
   - when both operands are character, short or integer data types, the result of the expression is an integer value.
   - when one of the operand is a long integer, the result is a long integer, unless one of the operand is an unsigned integer. In the later case, the other operand is converted to an unsigned integer value and the resulting value of the expression is an unsigned value.
2. **If any one operand is a floating point value:**
   - when one or both operands are floats, the result of the operation is a float value;
   - when one or both operands are doubles, the result of the operation is a double value;
   - when one or both operands are long doubles, the result of the operation is a long double value;

- **Note: Converting values to lower types can result in incorrect values. For example, the floating point value 4.5 gives the value 4 when it is converted to an integer value.**

Data types	
---	
long double	← highest type
double	
float	
unsigned long	
long int	
unsigned int	
int	
short int	
char	← lowest type

Determining Storage Size

- **C++ provides an operator for determining the amount of storage your compiler allocates for each data type. This operator is the *sizeof()* operator.**

- **Example:**

  sizeof(num1)

  sizeof(int)

  sizeof(float)

The item in parentheses can be a variable or a data type.

Example 2.4.1

```
#include <iostream.h>
int main()
{
   char c;
   short s;
   int i;
   long l;
   float f;
   double d;
   long double ld;
   cout << "sizeof c = " << sizeof(c)
        << "\tsizeof(char) = " << sizeof( char )
```

```
        << "\nsizeof s = " << sizeof(s)
        << "\tsizeof(short) = " << sizeof( short )
        << "\nsizeof i = " << sizeof (i)
        << "\tsizeof(int) = " << sizeof( int )
        << "\nsizeof l = " << sizeof(l)
        << "\tsizeof(long) = " << sizeof( long )
        << "\nsizeof f = " << sizeof (f)
        << "\tsizeof(float) = " << sizeof(float)
        << "\nsizeof d = " << sizeof (d)
        << "\tsizeof(double) = " << sizeof(double)
        << endl;
    return 0;
}
```

The output of the above program:

sizeof c = 1	**sizeof(char) = 1**
sizeof s = 2	**sizeof(short) = 2**
sizeof i = 4	**sizeof(int) = 4**
sizeof l = 4	**sizeof(long) = 4**
sizeof f = 4	**sizeof(float) = 4**
sizeof d = 8	**sizeof(double) = 8**

- **Example 3.5.1**

```
// this program calculates the circumference of a circle given its radius
#include <iostream.h>

int main()
{
  const float PI = 3.1416
  float radius, circumference;

  radius = 2.0;
  circumference = 2.0 * PI * radius;
  cout << "The circumference of the circle is "
       << circumference << endl;

  return 0;
}
```

The output of the above program:

```
The circumference of the circle is  12.5664
```

Summary

- A C/C++ program consists of many modules that can be functions or classes
- A function declaration includes a function header and a function body
- A function body contains many statements
- We can use variables and constants in statements
- Variables are used to store and manipulate data
- Constants are used to represent value
- Variables are associated with type