1

Introduction to Computers, the Internet and the Web

1.8 History of C

- **C**
  - **Evolved by Ritchie from two previous programming languages, BCPL and B**
  - **Used to develop UNIX**
  - **Used to write modern operating systems**
  - **Hardware independent (portable)**
  - **By late 1970's C had evolved to "Traditional C"**
- **Standardization**
  - **Many slight variations of C existed, and were incompatible**
  - **Committee formed to create a "unambiguous, machine-independent" definition**
  - **Standard created in 1989, updated in 1999**

Portability Tip 1.1

Because C is a hardware-independent, widely available language, applications written in C can run with little or no modifications on a wide range of different computer systems.

1.9 C Standard Library

- **C programs consist of pieces/modules called functions**
  - **A programmer can create his own functions**
    - **Advantage: the programmer knows exactly how it works**
    - **Disadvantage: time consuming**
  - **Programmers will often use the C library functions**
    - **Use these as building blocks**
  - **Avoid re-inventing the wheel**
    - **If a pre-made function exists, generally best to use it rather than write your own**
    - **Library functions carefully written, efficient, and portable**

5

Performance Tip 1.1

Using Standard C library functions instead of writing your own comparable versions can improve program performance, because these functions are carefully written to perform efficiently.

6

Portability Tip 1.2

Using Standard C library functions instead of writing your own comparable versions can improve program portability, because these functions are used in virtually all Standard C implementations.

1.10 C++

- **C++**
  - **Superset of C developed by Bjarne Stroustrup at Bell Labs**
  - **"Spruces up" C, and provides object-oriented capabilities**
  - **Dominant language in industry and academia**
- **Learning C++**
  - **Because C++ includes C, some feel it is best to master C, then learn C++**
  - **Starting in Chapter 18, we begin our introduction to C++**

1.14 Typical C Program Development Environment

- Phases of C++ Programs:
  - *Edit*
  - *Preprocess*
  - *Compile*
  - *Link*
  - *Load*
  - *Execute*

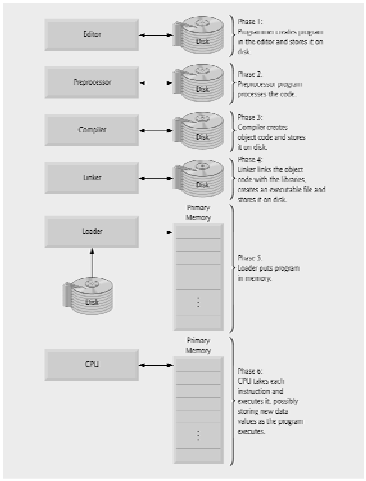


Fig. 1.1 | Typical C development environment.

9

Good Programming Practice 1.1

Write your C programs in a simple and straightforward manner. This is sometimes referred to as KIS ("keep it simple"). Do not "stretch" the language by trying bizarre usages.

10

Software Engineering Observation 1.2

Your computer and compiler are good teachers. If you are not sure how a feature of C works, write a sample program with that feature, compile and run the program and see what happens.

2

Introduction to C Programming

Outline

```
1  /* Fig. 2.1: fig02_01.c
2     A first program in C */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main( void )
7  {
8     printf( "Welcome to C!\n" );
9
10    return 0; /* indicate that program ended successfully */
11
12 } /* end function main */
```

```
Welcome to C!
```

`/*` and `*/` indicate comments – ignored by compiler

`#include` directive tells C to load a particular file

Left brace declares beginning of `main` function

Statement tells C to perform an action

`return` statement ends the function

Right brace declares end of `main` function

fig02_01.c

2.2 A Simple C Program: Printing a Line of Text

- `int main()`
  - C++ programs contain one or more functions, exactly one of which must be `main`
  - Parenthesis used to indicate a function
  - `int` means that `main` "returns" an integer value
  - Braces ({ and }) indicate a block
    - The bodies of all functions must be contained in braces

Good Programming Practice 2.1

Every function should be preceded by a comment describing the purpose of the function.

```
1  /* Fig. 2.1: fig02_01.c
2     A first program in C */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main( void )
7  {
8     printf( "Welcome to C!\n" );
9
10    return 0; /* indicate that program ended successfully */
11
12 } /* end function main */
```

```
Welcome to C!
```

2.2 A Simple C Program: Printing a Line of Text

- `printf( "Welcome to C!\n" );`
  - Instructs computer to perform an action
    - Specifically, prints the string of characters within quotes (" ")
  - Entire line called a statement
    - All statements must end with a semicolon (`;`)
  - Escape character (`\`)
    - Indicates that printf should do something out of the ordinary
    - `\n` is the newline character

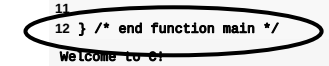
Escape sequence	Description
`\n`	Newline. Position the cursor at the beginning of the next line.
`\t`	Horizontal tab. Move the cursor to the next tab stop.
`\a`	Alert. Sound the system bell.
`\\`	Backslash. Insert a backslash character in a string.
`\"`	Double quote. Insert a double-quote character in a string.

Fig. 2.2 | Some common escape sequences.

Good Programming Practice 2.2

Add a comment to the line containing the right brace, `}`, that closes every function, including `main`.

```
1  /* Fig. 2.1: fig02_01.c
2     A first program in C */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main( void )
7  {
8     printf( "Welcome to C!\n" );
9
10    return 0; /* indicate that program ended successfully */
11
12 } /* end function main */
Welcome to C!
```



Good Programming Practice 2.3

The last character printed by a function that displays output should be a newline (`\n`). This ensures that the function will leave the screen cursor positioned at the beginning of a new line. Conventions of this nature encourage software reusability—a key goal in software development environments.

Outline

fig02_03.c

```
1  /* Fig. 2.3: fig02_03.c
2     Printing on one line with two printf statements */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main( void )
7  {
8     printf( "Welcome " );
9     printf( "to C!\n" );
10
11    return 0; /* indicate that program ended successfully */
12
13 } /* end function main */
```

printf statement starts printing from where the last statement ended, so the text is printed on one line.

```
Welcome to C!
```

Outline

fig02_04.c

```
1  /* Fig. 2.4: fig02_04.c
2     Printing multiple lines with a single printf */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main( void )
7  {
8     printf( "Welcome\nto\nC!\n" );
9
10    return 0; /* indicate that program ended successfully */
11
12 } /* end function main */
```

Newline characters move the cursor to the next line

```
Welcome
to
C!
```

Good Programming Practice 2.4

Indent the entire body of each function one level of indentation (we recommend three spaces) within the braces that define the body of the function. This indentation emphasizes the functional structure of programs and helps make programs easier to read.

Outline

fig02_03.c

```
1  /* Fig. 2.3: fig02_03.c
2     Printing on one line with two printf statements */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main( void )
7  {
8     printf( "Welcome " );
9     printf( "to C!\n" );
10
11    return 0; /* indicate that program ended successfully */
12
13 } /* end function main */
Welcome to C!
```

The tab key may be used to create indents, but tab stops may vary. We recommend using 2 or 3 spaces per level of indent.

Outline

fig02_05.c

```
1  /* Fig. 2.5: fig02_05.c
2     Addition program */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main( void )
7  {
8     int integer1; /* first number to be input by user  */
9     int integer2; /* second number to be input by user */
10    int sum;      /* variable in which sum will be stored */
11
12    printf( "Enter first integer\n" ); /* prompt */
13    scanf( "%d", &integer1 );          /* read an integer */
14
15    printf( "Enter second integer\n" ); /* prompt */
16    scanf( "%d", &integer2 );           /* read an integer */
17
18    sum = integer1 + integer2; /* assign total to sum */
19
20    printf( "Sum is %d\n", sum ); /* print sum */
21
22    return 0;  /* indicate that program ended successfully */
23
24 } /* end function main */
```

```
Enter first integer
45
Enter second integer
72
Sum is 117
```

Definitions of variables

scanf obtains a value from the user and assigns it to **integer1**

scanf obtains a value from the user and assigns it to **integer2**

Assigns a value to **sum**

2.3 Another Simple C Program: Adding Two Integers

- As before
  - Comments, `#include <stdio.h>` and `main`
- `int integer1, integer2, sum;`
  - Definition of variables
    - Variables: locations in memory where a value can be stored
  - `int` means the variables can hold integers (`-1`, `3`, `0`, `47`)
  - Variable names (identifiers)
    - `integer1`, `integer2`, `sum`
    - Identifiers: consist of letters, digits (cannot begin with a digit) and underscores(_)

      Case sensitive
  - Definitions appear before executable statements
    - If an executable statement references and undeclared variable it will produce a syntax (compiler) error

25

Common Programming Error 2.4

Using a capital letter where a lowercase letter should be used (for example, typing `Main` instead of `main`).

26

Good Programming Practice 2.6

Choosing meaningful variable names helps make a program self-documenting, i.e., fewer comments are needed.

Outline

fig02_05.c

```
1  /* Fig. 2.5: fig02_05.c
2     Addition program */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main( void )
7  {
8     int integer1; /* first number to be input by user  */
9     int integer2; /* second number to be input by user */
10    int sum;      /* variable in which sum will be stored */
11
12    printf( "Enter first integer\n" ); /* prompt */
13    scanf( "%d", &integer1 );          /* read an integer */
14
15    printf( "Enter second integer\n" ); /* prompt */
16    scanf( "%d", &integer2 );           /* read an integer */
17
18    sum = integer1 + integer2; /* assign total to sum */
19
20    printf( "Sum is %d\n", sum ); /* print sum */
21
22    return 0;  /* indicate that program ended successfully */
23
24 } /* end function main */
```

```
Enter first integer
45
Enter second integer
72
Sum is 117
```

Should be declared before executable statements in C. (Can be placed anywhere in C++)

Separate the definitions and executable statements with one blank line to emphasize where the definitions end and the executable statements begin.

2.3 Another Simple C Program: Adding Two Integers

- `scanf( "%d", &integer1 );`
  - Obtains a value from the user
    - `scanf` uses standard input (usually keyboard)
  - This `scanf` statement has two arguments
    - `%d` - indicates data should be a decimal integer
    - `&integer1` - location in memory to store variable
    - `&` is confusing in beginning – for now, just remember to include it with the variable name in `scanf` statements
  - When executing the program the user responds to the `scanf` statement by typing in a number, then pressing the *enter* (return) key

Good Programming Practice 2.10

Place a space after each comma (,) to make programs more readable.

```
5  /* function main begins program execution */
6  int main( void )
7  {
8     int integer1; /* first number to be input by user  */
9     int integer2; /* second number to be input by user */
10    int sum;      /* variable in which sum will be stored */
11
12    printf( "Enter first integer\n" ); /* prompt */
13    scanf( "%d", &integer1 );          /* read an integer */
14
15    printf( "Enter second integer\n" ); /* prompt */
16    scanf( "%d", &integer2 );           /* read an integer */
17
18    sum = integer1 + integer2; /* assign total to sum */
19
20    printf( "Sum is %d\n", sum ); /* print sum */
21
22    return 0;  /* indicate that program ended successfully */
23
24 } /* end function main */
```

2.3 Another Simple C Program: Adding Two Integers

- = (assignment operator)
  - Assigns a value to a variable
  - Is a binary operator (has two operands)

    ```
    sum = variable1 + variable2;
    sum gets variable1 + variable2;
    ```
  - Variable receiving value on left
- `printf( "Sum is %d\n", sum );`
  - Similar to `scanf`
    - `%d` means decimal integer will be printed
    - `sum` specifies what integer will be printed
  - Calculations can be performed inside `printf` statements

    ```
    printf( "Sum is %d\n", integer1 + integer2 );
    ```

Good Programming Practice 2.11

Place spaces on either side of a binary operator. This makes the operator stand out and makes the program more readable.

```
5  /* function main begins program execution */
6  int main( void )
7  {
8     int integer1; /* first number to be input by user  */
9     int integer2; /* second number to be input by user */
10    int sum;      /* variable in which sum will be stored */
11
12    printf( "Enter first integer\n" ); /* prompt */
13    scanf( "%d", &integer1 );          /* read an integer */
14
15    printf( "Enter second integer\n" ); /* prompt */
16    scanf( "%d", &integer2 );           /* read an integer */
17
18    sum = integer1 + integer2; /* assign total to sum */
19
20    printf( "Sum is %d\n", sum ); /* print sum */
21
22    return 0;  /* indicate that program ended successfully */
23
24 } /* end function main */
```

Common Programming Error 2.6

A calculation in an assignment statement must be on the right side of the = operator. It is a syntax error to place a calculation on the left side of an assignment operator.

a = b + c ; (O)

b + c = a ; (X)

Common Programming Errors (1/4)

2.7: Forgetting one or both of the double quotes surrounding the format control string in a `printf` or `scanf`.

2.8 Forgetting the `%` in a conversion specification in the format control string of a `printf` or `scanf`.

```
printf( "Sum is %d\n", sum );
```

Common Programming Errors (2/4)

2.9: Placing an escape sequence such as `\n` outside the format control string of a `printf` or `scanf`.

2.12: Placing inside the format control string the comma that is supposed to separate the format control string from the expressions to be printed.

```
printf( "Sum is %d\n", sum );
```

Common Programming Errors (3/4)

2.10: Forgetting to include the expressions whose values are to be printed in a `printf` containing conversion specifiers.

2.11: Not providing a conversion specifier when one is needed in a `printf` format control string to print the value of an expression.

```
printf( "Sum is %d\n", sum );
```

Common Programming Errors (4/4)

2.13: Forgetting to precede a variable in a `scanf` statement with an ampersand when that variable should, in fact, be preceded by an ampersand.

2.14: Preceding a variable included in a `printf` statement with an ampersand when, in fact, that variable should not be preceded by an ampersand.

```
scanf( "%d", &integer1 );
printf( "Sum is %d\n", sum );
```

2.4 Memory Concepts

- **Variables**
  - **Variable names correspond to locations in the computer's memory**
  - **Every variable has a name, a type, a size and a value**
  - **Whenever a new value is placed into a variable (through `scanf`, for example), it replaces (and destroys) the previous value**
  - **Reading variables from memory does not change them**

Fig. 2.6 | Memory location showing the name and value of a variable.

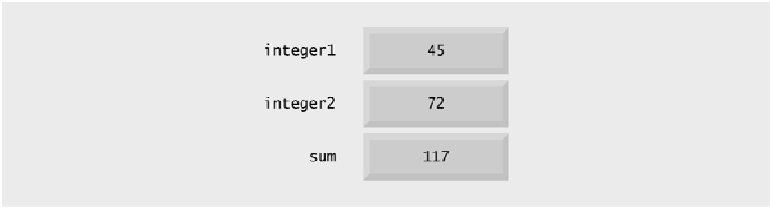


Fig. 2.8 | Memory locations after a calculation.

2.5 Arithmetic

- **Arithmetic calculations**
  - **Use `*` for multiplication and `/` for division**
  - **Integer division truncates remainder**
    - **`7 / 5` evaluates to `1`**
  - **Modulus operator(`%`) returns the remainder**
    - **`7 % 5` evaluates to `2`**
- **Operator precedence**
  - **Some arithmetic operators act before others (i.e., multiplication before addition)**
    - **Use parenthesis when needed**
  - **Example: Find the average of three variables `a`, `b` and `c`**
    - **Do not use: `a + b + c / 3`**
    - **Use: `(a + b + c ) / 3`**

Operator(s)	Operation(s)	Order of evaluation (precedence)
`( )`	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they are evaluated left to right.

`*`
`/`
`%`
	Multiplication
	Division
	Remainder
		Evaluated second. If there are several, they are evaluated left to right.

`+` `-`	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to right.

Fig. 2.10 | Precedence of arithmetic operators.

Good Programming Practice 2.12

Using redundant parentheses in complex arithmetic expressions can make the expressions clearer.

18

C++ as a Better C; Introducing Object Technology

Outline

```
1  // Fig. 18.1: fig18_01.cpp
2  // Addition program that displays the sum of two numbers.
3  #include <iostream> // allows program to perform input and output
4
5  int main()
6  {
7     int number1; // first integer to add
8
9     std::cout << "Enter first integer: "; // prompt user for data
10    std::cin >> number1; // read first integer from user into number1
11
12    int number2; // second integer to add
13    int sum; // sum of number1 and number2
14
15    std::cout << "Enter second integer: "; // prompt user for data
16    std::cin >> number2; // read second integer from user into number2
17    sum = number1 + number2; // add the numbers; store result in sum
18    std::cout << "Sum is " << sum << std::endl; // display sum; end line
19
20    return 0; // indicate that program ended successfully
21 } // end function main

Enter first integer: 45
Enter second integer: 72
Sum is 117
```

fig18_01.cpp

Include the contents of the `iostream`

Declare integer variables

Use stream extraction operator with standard input stream to obtain user input

Stream manipulator `std::endl` outputs a newline, then "flushes output buffer"

Concatenating, chaining or cascading stream insertion operations

18.3 A Simple Program: Adding Two Integers

- **C++ file names can have one of several extensions**
  - **Such as: .cpp, .cxx or .C (uppercase)**
- **Commenting**
  - **A // comment is a maximum of one line long**
  - **A /*…*/ C-style comments can be more than one line long**
- **`iostream`**
  - **Must be included for any program that outputs data to the screen or inputs data from the keyboard using C++-style stream input/output**
- **C++ requires you to specify the return type, possibly `void`, for all functions**
  - **Specifying a parameter list with empty parentheses is equivalent to specifying a `void` parameter list in C**

18.3 A Simple Program: Adding Two Integers (Cont.)

- **Declarations can be placed almost anywhere in a C++ program**
  - **They must appear before their corresponding variables are used in the program**
- **Input stream object**
  - **`std::cin` from `<iostream>`**
    - **Usually connected to keyboard**
    - **Stream extraction operator `>>`**
      - **Waits for user to input value, press *Enter* (Return) key**
      - **Stores value in variable to right of operator**
        - **Converts value to variable data type**
    - **Example**
      - `std::cin >> number1;`
        - **Reads an integer typed at the keyboard**
        - **Stores the integer in variable `number1`**

18.3 A Simple Program: Adding Two Integers (Cont.)

- **Stream manipulator `std::endl`**
  - **Outputs a newline**
  - **Flushes the output buffer**
- **The notation `std::cout` specifies that we are using a name (`cout`) that belongs to a "namespace" (`std`)**

18.3 A Simple Program: Adding Two Integers (Cont.)

- **Concatenating stream insertion operations**
  - **Use multiple stream insertion operators in a single statement**
    - **Stream insertion operation knows how to output each type of data**
  - **Also called chaining or cascading**
  - **Example**
    - `std::cout << "Sum is " << number1 + number2 << std::endl;`
      - **Outputs** `"Sum is "`
      - **Then, outputs sum of** `number1` **and** `number2`
      - **Then, outputs newline and flushes output buffer**

18.5 Header Files

- **C++ Standard Library header files**
  - **Each contains a portion of the Standard Library**
    - **Function prototypes for the related functions**
    - **Definitions of various class types and functions**
    - **Constants needed by those functions**
  - **"Instruct" the compiler on how to interface with library and user-written components**
  - **Header file names ending in** `.h`
    - **Are "old-style" header files**
    - **Superseded by the C++ Standard Library header files**
  - **Use** `#include` **directive to include class in a program**