SINGAPORE INSTITUTE OF TECHNOLOGY

ENG1008 C Programming

Topic 1

- Introduction to Programming Languages
- Introduction to C Programming

SINGAPORE INSTITUTE OF TECHNOLOGY

Introduction to Programming Languages

Programming Languages

SINGAPORE INSTITUTE OF TECHNOLOGY

- Programmers write instructions in various **programming languages**, some *directly understandable by computers* and others requiring *intermediate translation* steps.
- Any computer can directly understand only its own machine language, defined by its **hardware design**.
- **Machine languages** generally consist of numbers (ultimately reduced to **1s and 0s**). Such languages are cumbersome for humans.
- Programming in machine language—the numbers that computers could directly understand—was simply *too slow and tedious* for most programmers.
- Instead, they began using **English like abbreviations** to represent **elementary** operations.
- These abbreviations formed the basis of assembly languages.
- *Translator programs* called assemblers were developed to *convert assembly-language programs to machine language*.

Programming Languages

SINGAPORE INSTITUTE OF TECHNOLOGY

- Although assembly-language code is clearer to humans, it's incomprehensible to computers until translated to machine language.
- To **speed** the programming process even further, high-level languages were developed in which single statements could be written to accomplish *substantial tasks*.
- **High-level languages** allow you to **write instructions** that look almost like **everyday English** and contain commonly used **mathematical** expressions.
- *Translator programs* called compilers convert *high-level language programs into machine language.*
- Interpreter programs were developed to **execute** *high-level* language programs **directly**, although more **slowly** than compiled programs.
- Scripting languages such as JavaScript and PHP are processed by interpreters.

Performance Tip 1.1

Interpreters have an advantage over compilers in Internet scripting. An interpreted program can begin executing as soon as it's downloaded to the client's machine, without needing to be compiled before it can execute. On the downside, interpreted scripts generally run slower than compiled code.

The C Programming language

SINGAPORE INSTITUTE OF TECHNOLOGY

- **C** evolved from two previous languages, **BCPL** and **B**.
- **BCPL** was developed in 1967 by *Martin Richards* as a language for writing operating-systems software and compilers.
- *Ken Thompson* modeled many features in his **B** language after their counterparts in BCPL, and in 1970 he used B to create early versions of the UNIX operating system at Bell Laboratories.
- The **C language** was evolved from B by Dennis Ritchie at Bell Laboratories and was originally implemented in 1972.
- C initially became widely known as the **development language** of the *UNIX operating system*.
- Many of todays *leading operating systems* are written in C &/or C++
- C is **mostly hardware independent**.
- With careful design, it's possible to write C programs that are **portable** to most computers.

The C Programming language

SINGAPORE INSTITUTE OF TECHNOLOGY

Built for Performance

- C is widely used to develop systems that **demand performance**, such as operating systems, embedded systems, real-time systems and communications systems (Figure 1.5).

Application	Description
Operating systems	C's portability and performance make it desirable for implementing operating systems, such as Linux and portions of Microsoft's Windows and Google's Android. Apple's OS X is built in Objective-C, which was derived from C. We discuss some key popular desktop/notebook operating systems and mobile operating systems in Section 1.12.
Embedded systems	The vast majority of the microprocessors produced each year are embedded in devices other than general-purpose computers. These embedded systems include navigation systems, smart home appliances, home security systems, smartphones, robots, intelligent traffic intersections and more. C is one of the most popular programming languages for developing embedded systems, which typically need to run as fast as possible and conserve memory. For example, a car's anti-lock brakes must respond immediately to slow or stop the car without skidding; game controllers used for video games should respond instantaneously to prevent any lag between the controller and the action in the game, and to ensure smooth animations.

Fig. 1.5 | Some popular performance-oriented C applications.

The C Programming language

SINGAPORE INSTITUTE OF TECHNOLOGY

- By the late 1970s, C had evolved into what is now referred to as "*traditional C*." The publication in 1978 of Kernighan and Ritchie's book, *The C Programming Language,* drew wide attention to the language.
  - ***Standardization***
    - The rapid expansion of C over various types of computers (sometimes called hardware platforms) led to many variations that were similar but often incompatible.
    - In **1989**, the **C standard** was approved; this standard was updated in **1999** and is often referred to as **C99**.
  - ***The New C Standard***
    - The new C standard (referred to as **C11**) approved in 2011 and updated in 2018 (C18) refines and expands the capabilities of C.
    - Next update is likely to be released in 2022 [1].

Portability Tip 1.1

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

[1] "Programming Language C -C2x Charter." Accessed 28 August 2020. https://www.open-std.org/jtc1/sc22/wg14/www/docs/n2086.htm

C Standard Library

- Avoid reinventing the wheel.
- Instead, use existing pieces—this is called software reuse.
- When programming in C you'll typically use the following building blocks:
  - C Standard Library functions
  - Open-source C library functions
  - Functions you create yourself
  - Functions other people (whom you trust) have created and made available to you

C Standard Library

- The advantage of creating your own functions is that you'll know exactly how they work. You'll be able to examine the C code.
- The disadvantage is the time-consuming effort that goes into designing, developing and debugging

Performance Tip 1.2

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.

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.

Other languages: C++

- C++ was developed by Bjarne Stroustrup at Bell Laboratories.
- It has its roots in C, providing a number of features that "spruce up" the C language.
- More important, it provides capabilities for object-oriented programming.
- Objects are essentially **reusable** software components that model items in the real world.
- Using a **modular, object-oriented design** and implementation approach can make software development groups more productive.

Other languages: Python

SINGAPORE INSTITUTE OF TECHNOLOGY

- Python is an object-oriented language, developed by Guido van Rossum and was released in 1991.
- Rapidly became popular for educational and scientific computing due to the following reasons:
  - Open-source, free and widely available
  - Supported by a massive open-source community
  - Relatively easy to learn
  - Code is easier to read than many other popular programming languages
  - Developer productivity is enhanced with extensive standard libraries and thousands of third-party open-source libraries
  - Popular in web development, AI and data science, which are currently hot areas of growth, and also financial community.

C Programming Development

SINGAPORE INSTITUTE OF TECHNOLOGY

- C systems generally consist of several parts: a **program development environment**, the **language** and the **C Standard Library**.
- C programs typically go through six phases to be executed (Fig. 1.7).
- These are: edit, preprocess, compile, link, load and execute.
- Although *C How to Program, 9/e* is a generic C textbook (written independently of the details of any particular operating system), we concentrate in this section on a typical Linux-based C system.

Phase 1 Creating the program

SINGAPORE INSTITUTE OF TECHNOLOGY

- Phase 1 consists of **editing** a file.
- This is accomplished with an editor program.
- Two editors widely used on Linux systems are vi and emacs.
- Software packages for the C/C++ integrated program development environments such as Eclipse, Microsoft Visual Studio and Code:Blocks have editors that are integrated into the programming environment.
- You type a C program with the editor, make corrections if necessary, then store the program on a secondary storage device such as a hard disk.
- C program file names should end with the **.c** extension

Phases 2 and 3: Preprocessing and Compiling

SINGAPORE INSTITUTE OF TECHNOLOGY

- In **Phase 2**, you give the command to compile the program.
- The compiler translates the C program into machine language-code (also referred to as object code).
- In a C system, a preprocessor program executes automatically before the compiler's translation phase begins.
- The C preprocessor obeys special commands called preprocessor directives, which indicate that certain manipulations are to be performed on the program before compilation.
- These manipulations usually consist of including other files in the file to be compiled and performing various text replacements.

Phases 2 and 3: Preprocessing and Compiling

SINGAPORE INSTITUTE OF TECHNOLOGY

- In **Phase 3**, the **compiler translates** the C program into **machine-language code**.
- A syntax error occurs when the compiler cannot recognize a statement because it violates the rules of the language.
- Syntax errors are also called compile errors, or compile-time errors.

Phase 4 Linking

- The next phase is called linking.
- C programs typically contain **references to functions** defined elsewhere, such as in the **standard libraries** or in the **private libraries** of groups of programmers working on a particular project.
- The object code produced by the C compiler typically contains "holes" due to these missing parts.
- A linker links the **object code** with the code for the missing functions to produce an executable image (with no missing pieces).
- On a typical Linux system, the command to compile and link a program is called gcc (the GNU compiler).

Phase 4 Linking

- To compile and link a program named welcome.c type
  - `gcc welcome.c`
- at the Linux prompt and press the *Enter* key (or *Return* key).
- Note: Linux commands are case sensitive; make sure that each c is lowercase and that the letters in the filename are in the appropriate case.
- If the program compiles and links correctly, a file called a.out is produced. In Windows, a.exe is produced
- This is the executable image of our welcome.c program.
- To specify the output filename, use the '-o' flag and type
  - `gcc welcome.c -o welcome`
  - this will produce a welcome.out file in Linux or welcome.exe file in Windows

Phase 5 Loading

SINGAPORE INSTITUTE OF TECHNOLOGY

- The next phase is called loading.
- Before a program can be executed, the program must first be placed in **memory**.
- This is done by the loader, which takes the executable image from **disk** and **transfers** it to **memory**.
- Additional components from **shared libraries** that support the program are also **loaded**.

Phase 6 Executing

- Finally, the computer, under the control of its CPU, executes the program one instruction at a time.
- To **load and execute** the program on a Linux system, type `./a.out` at the Linux prompt and press *Enter.* On a Windows system, type a and press *Enter.*

Phases 1 to 4

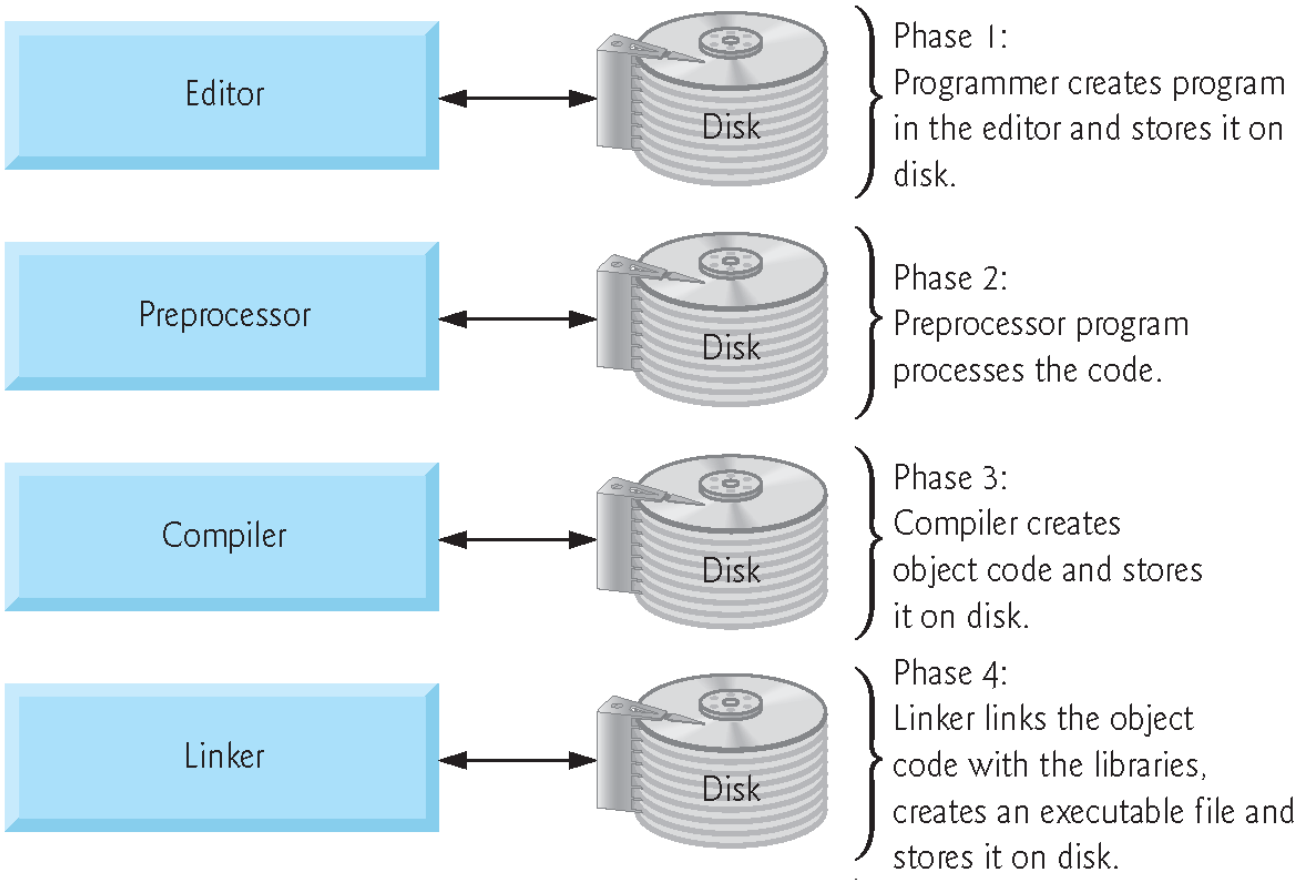


Fig. 1.7 | Typical C development environment. (Part 1 of 3.)

Phases 5 & 6

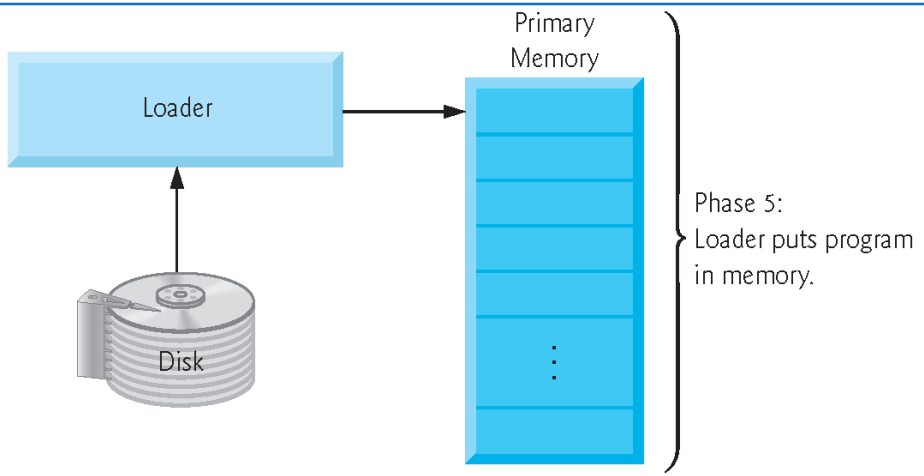


Fig. 1.7 | Typical C development environment. (Part 2 of 3.)

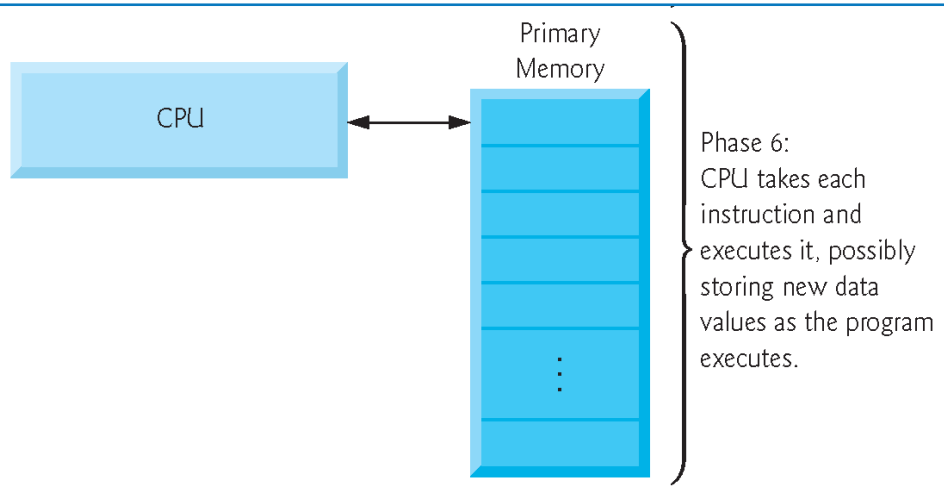


Fig. 1.7 | Typical C development environment. (Part 3 of 3.)

Problems during execution

- Programs do not always work on the first try.
- Each of the preceding phases can fail because of various errors that we'll discuss.
- For example, an executing program might attempt to *divide by zero* (an illegal operation on computers just as in arithmetic).
- This would cause the computer to display an error message.
- You would then return to the edit phase, make the necessary corrections and proceed through the remaining phases again to determine that the corrections work properly.

Common Programming Error 1.1

Errors such as division-by-zero occur as a program runs, so they are called runtime errors or execution-time errors. Divide-by-zero is generally a fatal error, i.e., one that causes the program to terminate immediately without successfully performing its job. Nonfatal errors allow programs to run to completion, often producing incorrect results.

Standard Input, Output and Error

SINGAPORE INSTITUTE OF TECHNOLOGY

- Most C programs **input** and/or **output** data.
- Certain C functions take their input from stdin (the standard input stream), which is normally the **keyboard**, but stdin can be connected to another stream.
- Data is often output to stdout (the standard output stream), which is normally the computer screen, but stdout can be connected to another stream.
- When we say that a program prints a result, we normally mean that the result is displayed on a **screen**.
- Data may be output to devices such as disks and printers.

Standard Input, Output and Error

SINGAPORE INSTITUTE OF TECHNOLOGY

- There is also a standard error stream referred to as stderr.
- The stderr stream (normally connected to the screen) is used for displaying **error messages**.
- It's common to route regular output data, i.e., stdout, to a device other than the screen while keeping stderr assigned to the screen so that the user can be immediately informed of errors.

SINGAPORE INSTITUTE OF TECHNOLOGY

Introduction to C Programming

Objectives

- To write a simple C program
- To use basic data types and variables
- To learn about memory concepts
- To perform basic Input from keyboard / Output to screen
- To perform C Arithmetic operations
- C's keywords

A simple C program

SINGAPORE INSTITUTE OF TECHNOLOGY

- Simple C program to print a line of text

```
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  } // end function main
```

```
Welcome to C!
```

Fig. 2.1 | A first program in C.

A simple C program

- **Comments**
  - Insert comments for documentation
    - improves readability
    - helps other people to read and understand your code
  - Begins with // or to use /* … */
  - Ignored by the C compiler
    - machine object code is not generated for it
    - will not perform any action when the program is run
- **#include <stdio.h>**
  - a **directive** to the C **pre-processor**
  - processed by the C pre-processor before compilation
  - to include the contents of the standard input/output header; <stdio.h> shows standard I/O operations

A simple C program

- **Blank lines and White spaces**
  - blank lines, space (characters), tab (characters)/indent
    - improves readability
  - these are known as white space
  - normally ignored by the compiler

- **main** function – int main(void)
  - (main) part of every C program; one of the functions
  - every C program *begins execution* at the function main
  - keyword *int* to the left of main indicates that the main function will return an integer value
  - the *void* in parentheses indicates that main does not receive any data

A simple C program

- **{ }**
  - left brace { begins the body of every function
  - the corresponding right brace } ends each function
  - code between the pair of braces is a **block**

- **Output** statement
  - printf(“Welcome to C!\n”);
  - causes the computer to perform the print/output action
  - The string “Welcome to C!” is printed followed by the newline \n
  - The blackslash \ is called an escape character

Escape sequence

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. Produces a sound or visible alert without changing the current cursor position.
\\	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 .

Linker and Executables

- **Linker**
  - **standard library functions** (e.g. printf, scantf) are not part of the C programming language
  - the C compiler does not know where the standard library functions are
  - the linker ***locates the library functions*** and ***inserts*** the correct ***calls*** to the library functions in the **object code**
- **Executables**
  - the object program is complete and can be executed
  - To load and execute the executables
    - ./a or ./programname

Basic Data Types

- **Basic data types**
  - **int** (integer, a whole number)
    - Size 4 bytes
  - **float** (floating point, real numbers, has decimal point)
    - Size 4 bytes
  - **double** (double precision floating point number)
    - Size 8 bytes
  - **char** (character)
    - Size 1 byte
  - **sizeof** operator
- **unsigned and signed**
- **void**

Check out code at
http://tpcg.io/_WHDQ7Z

Variables and Memory concepts

SINGAPORE INSTITUTE OF TECHNOLOGY

- **Variables**
  - Variables are created with names that are associated with memory locations in the computer system
  - Each variable is defined with a **name** and **data type**, and has a **value** and **location in the computer's memory**
  - When a new value is placed into a variable, it will replace the previous value
  - Reading variables in memory is non-destructive – it will not change the value stored in it
- Variable **name** – valid Identifiers and case sensitivity
  - an identifier consists of a series of characters (letter, digits and underscores; does not begin with a digit)
  - C is **case sensitive**

Basic I/O

- **Output to screen**
  - standard library function **printf**
  - e.g. printf(“Hello World\n”);
  - e.g. int a = 1;
    printf(“Today is Day %d\n”, a);
- **Input from keyboard**
  - Standard library function **scanf**
  - e.g. int b;
    scanf(“%d”,&b);

  first part - %d : is the format control string

  second part - &b : provides to scanf the address in memory of the variable b

Arithmetic in C

- **C Arithmetic operators** (ref Figure 2.9)
  - Addition +, Subtraction -, Multiplication *
  - Division / and Reminder or mod %
  - What is 7/4 ? What is 7%4

7.0/4 = 1.75
Dividing real number with integer :. get decimal place

1 3

C operation	Arithmetic operator	Algebraic expression	C expression
Addition	+	$f + 7$	`f + 7`
Subtraction	–	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`
Remainder	%	$r \bmod s$	`r % s`

Fig. 2.9 | Arithmetic operators.

- **Division by 0** error

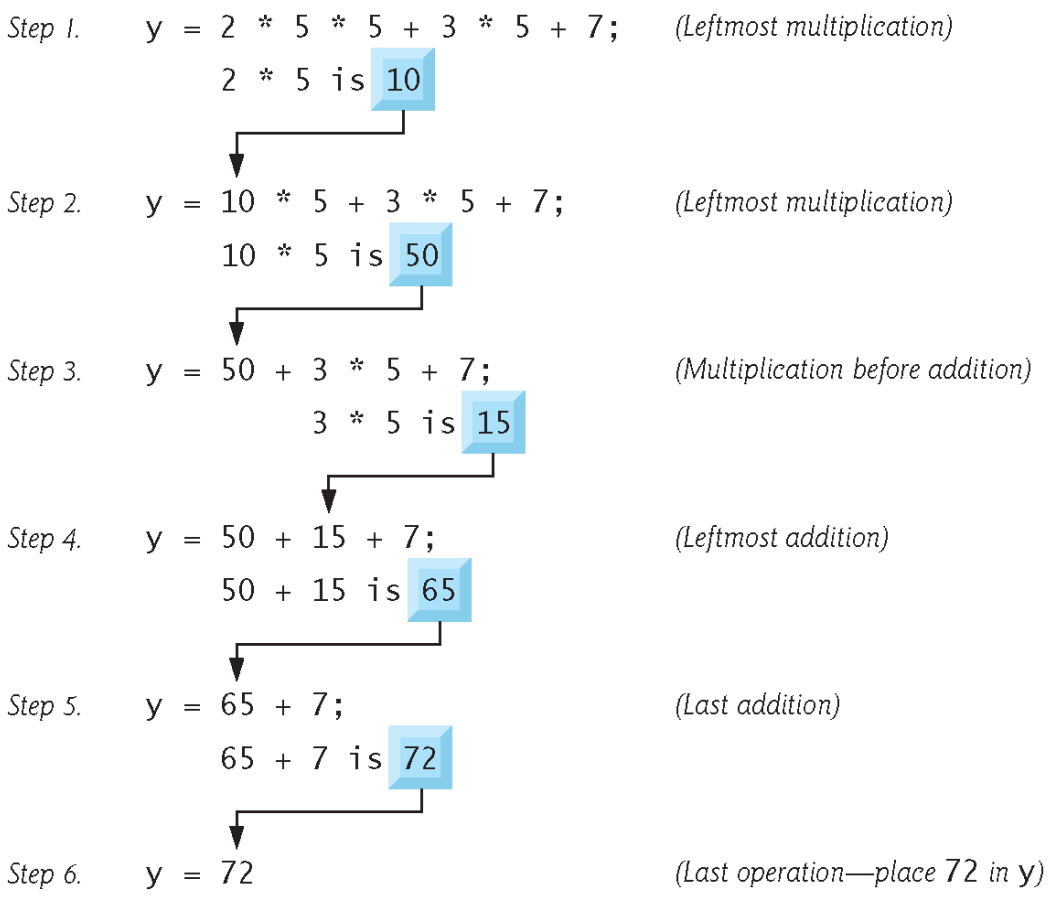
Arithmetic in C

- **Parentheses** for grouping subexpressions
  - e.g. (a * b) / (c + d)
- **Rules of Operator Precedence**
  - similar rules to algebra
  - parentheses has the highest level of precedence

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're evaluated left to right.
*	Multiplication	Evaluated second. If there are several, they're evaluated left to right.
/	Division	
%	Remainder	
+	Addition	Evaluated third. If there are several, they're evaluated left to right.
-	Subtraction	
=	Assignment	Evaluated last.

Fig. 2.10 | Precedence of arithmetic operators.

Arithmetic in C



y = 2*5*(5+3)*5+7;

Fig. 2.11 | Order in which a second-degree polynomial is evaluated.

Addition C program

```
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 entered by user
9     int integer2; // second number to be entered 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 } // end function main
```

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

Fig. 2.5 | Addition program. (Part 2 of 2.)

Equality and Relational Operators

SINGAPORE INSTITUTE OF TECHNOLOGY

Algebraic equality or relational operator	C equality or relational operator	Example of C condition	Meaning of C condition
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

Fig. 2.12 | Equality and relational operators.

C keywords

do not use this keywords as variables

Keywords			
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Keywords added in C99 standard

```
_Bool  _Complex  _Imaginary  inline  restrict
```

Keywords added in C11 draft standard

```
_Alignas  _Alignof  _Atomic  _Generic  _Noreturn  _Static_assert  _Thread_local
```

Fig. 2.15 | C's keywords.

Do not use these keywords as identifiers as they have special meaning to the compiler