Fundamentals of Programming
Session 5

Instructor: Reza Entezari-Maleki

Email: entezari@ce.sharif.edu
Outlines

- Memory Concepts
- Arithmetic in C
- Algorithms
- Pseudocode
- Control Structures
- The if Selection Statement
Memory Concepts

- Variable names such as `integer1`, `integer2` and `sum` actually correspond to locations in the computer’s memory.

- Every variable has a name, a type and a value.

- In the addition program of Fig. 2.5, when the statement (line 13)
  
  ```c
  scanf( "%d", &integer1 ); /* read an integer */
  ```

  is executed, the value typed by the user is placed into a memory location to which the name `integer1` has been assigned.

- Suppose the user enters the number 45 as the value for `integer1`.

- The computer will place 45 into location `integer1`. 
Whenever a value is placed in a memory location, the value replaces the previous value in that location; thus, placing a new value into a memory location is said to be destructive.

Returning to our addition program again, when the statement (line 16)
- `scanf( "%d", &integer2 ); /* read an integer */`
executes, suppose the user enters the value 72.

This value is placed into location integer2.

These locations are not necessarily adjacent in memory.
Memory Concepts …

- Once the program has obtained values for integer1 and integer2, it adds these values and places the sum into variable sum.
- The statement (line 18)
  - `sum = integer1 + integer2; /* assign total to sum */`
  - that performs the addition also replaces whatever value was stored in sum.
- This occurs when the calculated sum of integer1 and integer2 is placed into location sum (destroying the value already in sum).
- When a value is read from a memory location, the process is said to be nondestructive.
Arithmetic in C

- The C arithmetic operators are summarized in Fig. 2.9.
- Note the use of various special symbols not used in algebra.
- The asterisk (*) indicates multiplication and the percent sign (%) denotes the remainder operator, which is introduced below.
- In algebra, if we want to multiply a times b, we can simply place these single-letter variable names side by side as in ab.
- In C, however, if we were to do this, ab would be interpreted as a single, two-letter name (or identifier).
- Therefore, C requires that multiplication be explicitly denoted by using the * operator as in a * b.
## Arithmetic in C …

<table>
<thead>
<tr>
<th>C operation</th>
<th>Arithmetic operator</th>
<th>Algebraic expression</th>
<th>C expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>+</td>
<td>( f + 7 )</td>
<td>( f + 7 )</td>
</tr>
<tr>
<td>Subtraction</td>
<td>−</td>
<td>( p - c )</td>
<td>( p - c )</td>
</tr>
<tr>
<td>Multiplication</td>
<td>*</td>
<td>( bm )</td>
<td>( b \times m )</td>
</tr>
<tr>
<td>Division</td>
<td>(/)</td>
<td>( x / y ) or ( x \div y ) \ or ( x + y )</td>
<td>( x / y )</td>
</tr>
<tr>
<td>Remainder</td>
<td>(%)</td>
<td>( r \mod s )</td>
<td>( r % s )</td>
</tr>
</tbody>
</table>

**Fig. 2.9**  | Arithmetic operators.
The arithmetic operators are all binary operators.

For example, the expression 3 + 7 contains the binary operator + and the operands 3 and 7.

Integer division yields an integer result.

For example, the expression 7 / 4 evaluates to 1 and the expression 17 / 5 evaluates to 3.

C provides the remainder operator, %, which yields the remainder after integer division.

The remainder operator is an integer operator that can be used only with integer operands.

The expression x % y yields the remainder after x is divided by y.

Thus, 7 % 4 yields 3 and 17 % 5 yields 2.
Common Programming Error 2.16

An attempt to divide by zero is normally undefined on computer systems and generally results in a fatal error, i.e., an error that causes the program to terminate immediately without having successfully performed its job. Nonfatal errors allow programs to run to completion, often producing incorrect results.
Arithmetic in C …

- Arithmetic expressions in C must be written in straight-line form to facilitate entering programs into the computer.
- Thus, expressions such as “a divided by b” must be written as \( a/b \) so that all operators and operands appear in a straight line.
- Parentheses are used in C expressions in the same manner as in algebraic expressions.
- For example, to multiply a times the quantity \( b + c \) we write \( a \times (b + c) \).
Arithmetic in C …

- C applies the operators in arithmetic expressions in a precise sequence determined by the following rules of operator precedence, which are generally the same as those in algebra:
  - Operators in expressions contained within pairs of parentheses are evaluated first. Thus, parentheses may be used to force the order of evaluation to occur in any sequence you desire. Parentheses are said to be at the "highest level of precedence." In cases of nested, or embedded, parentheses, such as
    - \(( (a + b) + c)\)
  - the operators in the innermost pair of parentheses are applied first.
Multiplication, division and remainder operations are applied first. If an expression contains several multiplication, division and remainder operations, evaluation proceeds from left to right. Multiplication, division and remainder are said to be on the same level of precedence.

Addition and subtraction operations are evaluated next. If an expression contains several addition and subtraction operations, evaluation proceeds from left to right. Addition and subtraction also have the same level of precedence, which is lower than the precedence of the multiplication, division and remainder operations.
### Arithmetic in C ...

<table>
<thead>
<tr>
<th>Operator(s)</th>
<th>Operation(s)</th>
<th>Order of evaluation (precedence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>Parentheses</td>
<td>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.</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>Evaluated second. If there are several, they’re evaluated left to right.</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td></td>
</tr>
<tr>
<td>%</td>
<td>Remainder</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>Addition</td>
<td>Evaluated last. If there are several, they’re evaluated left to right.</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td></td>
</tr>
</tbody>
</table>

**Fig. 2.10** | Precedence of arithmetic operators.
Arithmetic in C …

**Fig. 2.11** Order in which a second-degree polynomial is evaluated.
Arithmetic in C …

- As in algebra, it is acceptable to place unnecessary parentheses in an expression to make the expression clearer.
- These are called redundant parentheses.
Algorithms

- Before writing a program to solve a particular problem, it’s essential to have a thorough understanding of the problem and a carefully planned approach to solving the problem.

- The solution to any computing problem involves executing a series of actions in a specific order.

- A procedure for solving a problem in terms of
  - the actions to be executed, and
  - the order in which these actions are to be executed

- is called an algorithm.
Algorithms …

- Correctly specifying the order in which the actions are to be executed is important.
- Specifying the order in which statements are to be executed in a computer program is called **program control**.
Pseudocode

- **Pseudocode** is an artificial and informal language that helps you develop algorithms.

- Pseudocode is similar to everyday English; it’s convenient and user friendly although it’s not an actual computer programming language.

- Pseudocode programs are not executed on computers.

- Rather, they merely help you “think out” a program before attempting to write it in a programming language such as C.
Pseudocode …

- A carefully prepared pseudocode program may be converted easily to a corresponding C program.

- Pseudocode consists only of action statements—those that are executed when the program has been converted from pseudocode to C and is run in C.

- Definitions are not executable statements. They are messages to the compiler.
Pseudocode …

- For example, the definition
  - `int i;`
- simply tells the compiler the type of variable `i` and instructs the compiler to reserve space in memory for the variable.
- But this definition does not cause any action—such as input, output, or a calculation—to occur when the program is executed.
- Some programmers choose to list each variable and briefly mention the purpose of each at the beginning of a pseudocode program.
Control Structures

- Normally, statements in a program are executed one after the other in the order in which they’re written. This is called sequential execution.

- Various C statements we’ll soon discuss enable you to specify that the next statement to be executed may be other than the next one in sequence. This is called transfer of control.

- The goto statement allows programmers to specify a transfer of control to one of many possible destinations in a program.

- The notion of so-called structured programming became almost synonymous with “goto elimination.”

- Research had demonstrated that programs could be written without any goto statements.
Control Structures …

• The results were impressive, as software development groups reported reduced development times, more frequent on-time delivery of systems and more frequent within-budget completion of software projects.

• Programs produced with structured techniques were clearer, easier to debug and modify and more likely to be bug free in the first place.

• Research had demonstrated that all programs could be written in terms of only three control structures, namely the sequence structure, the selection structure and the repetition structure.
Control Structures …

- The sequence structure is built into C.
- Unless directed otherwise, the computer executes C statements one after the other in the order in which they’re written.
- The flowchart segment of Fig. 3.1 illustrates C’s sequence structure.
- A flowchart is a graphical representation of an algorithm or of a portion of an algorithm.
- Flowcharts are drawn using certain special-purpose symbols such as rectangles, diamonds, ovals, and small circles; these symbols are connected by arrows called flowlines.
Control Structures …

**Fig. 3.1** | Flowcharting C’s sequence structure.
Like pseudocode, flowcharts are useful for developing and representing algorithms, although pseudocode is preferred by most programmers.

Consider the flowchart for the sequence structure in Fig. 3.1.

We use the rectangle symbol, also called the action symbol, to indicate any type of action including a calculation or an input/output operation.

When drawing a flowchart that represents a complete algorithm, an oval symbol containing the word “Begin” is the first symbol used in the flowchart; an oval symbol containing the word “End” is the last symbol used.

When drawing only a portion of an algorithm as in Fig. 3.1, the oval symbols are omitted in favor of using small circle symbols, also called connector symbols.

Perhaps the most important flowcharting symbol is the diamond symbol, also called the decision symbol, which indicates that a decision is to be made.
C provides three types of selection structures in the form of statements.

The `if` selection statement (called a *single-selection statement*) either performs (selects) an action if a condition is true or skips the action if the condition is false.

The `if...else` selection statement (double-selection statement) performs an action if a condition is true and performs a different action if the condition is false.

The `switch` selection statement (multiple-selection statement) performs one of many different actions depending on the value of an expression.

C provides three types of repetition structures in the form of statements, namely `while`, `do...while`, and `for`. 
Control Structures …

- C has only seven control statements: sequence, three types of selection and three types of repetition.

- Each C program is formed by combining as many of each type of control statement as is appropriate for the algorithm the program implements.

- As with the sequence structure of Fig. 3.1, we’ll see that the flowchart representation of each control statement has two small circle symbols, one at the entry point to the control statement and one at the exit point.

- These single-entry/single-exit control statements make it easy to build programs.

- The control-statement flowchart segments can be attached to one another by connecting the exit point of one control statement to the entry point of the next.
The `if` Selection Statement

- Figure 2.13 uses six `if` statements to compare two numbers input by the user.
- If the condition in any of these `if` statements is true, the `printf` statement associated with that `if` executes.
# The if Selection Statement ... 

```c
/* Fig. 2.13: fig02_13.c */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
    int num1; /* first number to be read from user */
    int num2; /* second number to be read from user */

    printf( "Enter two integers, and I will tell you\n" );
    printf( "the relationships they satisfy: " );

    scanf( "%d%d", &num1, &num2 ); /* read two integers */

    if ( num1 == num2 ) {
        printf( "%d is equal to %d\n", num1, num2 );
    } /* end if */

    if ( num1 != num2 ) {
        printf( "%d is not equal to %d\n", num1, num2 );
    } /* end if */
```

**Fig. 2.13** Using if statements, relational operators, and equality operators. (Part I of 3.)
The if Selection Statement ...

```c
24    if ( num1 < num2 ) {
25        printf( "%d is less than %d\n", num1, num2 );
26    } /* end if */
27
28    if ( num1 > num2 ) {
29        printf( "%d is greater than %d\n", num1, num2 );
30    } /* end if */
31
32    if ( num1 <= num2 ) {
33        printf( "%d is less than or equal to %d\n", num1, num2 );
34    } /* end if */
35
36    if ( num1 >= num2 ) {
37        printf( "%d is greater than or equal to %d\n", num1, num2 );
38    } /* end if */
39
40    return 0; /* indicate that program ended successfully */
41 } /* end function main */
```

Fig. 2.13 | Using if statements, relational operators, and equality operators. (Part 2 of 3.)
The **if** Selection Statement …

<table>
<thead>
<tr>
<th>Enter two integers, and I will tell you the relationships they satisfy:</th>
<th>3 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 is not equal to 7</td>
<td></td>
</tr>
<tr>
<td>3 is less than 7</td>
<td></td>
</tr>
<tr>
<td>3 is less than or equal to 7</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Enter two integers, and I will tell you the relationships they satisfy:</th>
<th>12 12</th>
</tr>
</thead>
<tbody>
<tr>
<td>22 is not equal to 12</td>
<td></td>
</tr>
<tr>
<td>22 is greater than 12</td>
<td></td>
</tr>
<tr>
<td>22 is greater than or equal to 12</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Enter two integers, and I will tell you the relationships they satisfy:</th>
<th>7 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 is equal to 7</td>
<td></td>
</tr>
<tr>
<td>7 is less than or equal to 7</td>
<td></td>
</tr>
<tr>
<td>7 is greater than or equal to 7</td>
<td></td>
</tr>
</tbody>
</table>

*Fig. 2.13*  | Using **if** statements, relational operators, and equality operators. (Part 3 of 3.)