Agenda

C uses some notations that may appear strange to people who have not programmed computers. We begin by considering a simple C program. Our first example prints a line of text. The program is shown in Fig. 1.1.

```
1   // A first program in C.
2   #include <stdio.h>
3
4   //function main begins program execution
5   int main( void )
6   {
7      printf( "Welcome to C!\n" );
8   } // end function main
```

Fig. 1.1 | A first program in C.

- Even though this program is simple, it illustrates several important features of the C language. Line 1 begins with //, indicating that this line is comment. You insert comments to document programs and improve program readibility.

- Comments do not cause the computer to perform any action when the program is run. Comments are ignored by the C compiler and do not cause any machine-language object code to be generated.

- You can also use /* */ multi-line comments in which everything from /* on the first line to */ at the end of the last line is a comment.

```
1    // A first program in C.
2    #include <stdio.h>
3
4    //function main begins program execution
5    int main( void )
6    {
7       printf( "Welcome to C!\n" );
8    } // end function main
```

Fig. 1.1 | A first program in C.

Line 2 #include <stdio.h> is a direction to the C preprocessor. Lines beginning with # are processed by the preprocessor before compilation. Line 2 tells the preprocessor to include the contents of the standart input/output header (<stdio.h>) in the program. This header contains information used by the compiler when compiling calls to standard input/output library functions such as printf.

```
1    // A first program in C.
2    #include <stdio.h>
3
4    //function main begins program execution
5    int main( void )
6    {
7       printf( “Welcome to C!\n” );
8    } // end function main
```

Fig. 1.1 | A first program in C.

- Line 3 is simply a blank line. You use blank lines, space characters and tab characters to make programs easier to read. Together, these characters are known as white space. White-space characters are normally igrnoed by the compiler.

```
1   // A first program in C.
2   #include <stdio.h>
3
4   //function main begins program execution
5   int main( void )
6   {
7      printf( "Welcome to C!\n" );
8   } // end function main
```

Fig. 1.1 | A first program in C.

- Line 6 int main(void) is a part of every C program. The parentheses after main indicated that main is a program building block called a function. C programs contain one or more functions, one of which must be main.

- Every program in C begnis executing at the function main. Functions can return information. The keyword into the left of the main indicates that main returns an integer (whole-number) value.

- Functions also can receive information when they're called upon to execute. The void in parentheses here means that main does not any information.

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

Good Programming Practice
Indent the entire body of each function one level of indentation within the braces that define the body of the function. This indentation emphasizes the functional structure of program and helps make programs easier to read.

- A left brace, {, begins the body of every function. A corresponding right brace ends each function. This pairs of braces and the portion of the program between the braces is called a block. The block is an important program unit in C.

- Line 7 printf(“Welcome to C!\n”); instructs the computer to perform an action, namely to pring on the screen the string of characters marked by the quatation marks. A string is sometimes called a character string, a message or a literal.

- The entire line, including the printf function (the “f” stands for “formatted”), its argument within the parentheses and the semicolon (;), is called statement. Every statement must end with a semicolon (also known as the statement terminatior).

- Functions also can receive information when they’re called upon to execute. The void in parentheses here means that main does not any information.

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

```
1   // A first program in C.
2   #include <stdio.h>
3
4   //function main begins program execution
5   int main( void )
6   {
7      printf( “Welcome to C!\n” );
8   } // end function main
```

Fig. 1.1 | A first program in C.

Notice that the characters \n were not printed on the scree. The backslash (\) is called an espace character. It indicates that printf is supposed to do something out of the ordinary. When encountering a backslash in a string, the compiler looks ahead at the next character ad combines it with the backslash to form an escape sequence.

The escape sequence \n means newline. When a newline appears in the string output by a printf, the newline causes the cursor to position to the beginning of the next line on the screen. Some common escape sequences are listed in Fig. 1.2.

Escape sequence	Description
\n	Newline. Position the cursor at the beginning of the next line.
\t	Horizantal 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 backalash character in a string.
\”	Double quote. Insert a double-quote character in a string.

Fig. 1.2 | Some common escape sequences.

- Standart library functions like printf and scanf are not part of the C programming language. For example, the compiler cannot find a spelling error in printf or scanf. When the compiler compiles a printf statement, it merely provides space in the object program for a “call” to the library function

- But the compiler does not know where the library functions are--the linker does. When the linker runs, it locates the library functions and inserts the proper calls to those library functions in the object program.

- For this reason, the linked program is called an executable. If the function name is misspelled, the linker will spot the error, because it will not be able to match the name in the C program with the name of any known function in the libraries.

Another Simple C Program: Adding Two Integers

- Our next program uses the Standart Library function scanf to obtain two integers types by a user at the keyboard, computes the sum of these values and prints the result usnig printf. The program is shown in Fig. 1.3.

```
1   // Addition program.
2   #include <stdio.h>
3
4   function main begins program execution
5   int main( void )
6   {
7   int integer1; // first number to be read from user
8   int integer2; // second number to be rad from user
9   int sum; // variable in which sum will be stored
10
11  printf( “Enter first integer\n” ); // prompt
12  scanf( “%d”, &integer1 ); // read an integer
13
14  printf( “Enter second integer\n” ); // prompt
15  scanf( “%d”, &integer2 ); // read an integer
16
17  sum = integer1 + integer2 // assign total to sum
18
19  printf( “Sum is %d\n”, sum  );
20  } // end function main
```

Fig 1.3 | Addition Program

- Lines 7-9 are definitions. The names integer1, integer2 and sum are the names of variables--locations in memory where values can be stored for use by a program. These definitions specify that varaible integer1, integer2 and sum are of type int, which means that they'll hold integer values, i.e., whole numbers such as 7, -11, 0, 31914 and the like.

- All variables must be defined with a name and a data type before they can be used in a program. The C standard allows you to place each variable definition anywhere in main before that variable's first use in the code. Some compilers, such as GNU gcc, have implemented this capability.

- The preceeding definitions could have been combined into a single definition statement as follows;

```
int integer1, integer2, sum;
```

but that would have made it difficult to describe the variables with corresponding comments as we did in lines 7-9.

- A variable name in C is any valid indentifier. An identifier is a series of characters consisting of letters, digits and underscores (_) that does not begin with a digit. C is case sensitive--uppercase and lowercase letters are different in C, so a1 and A1 are different identifiers.

Good Programming Practices

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

Good Programming Practices

The first letter of an identifier used as a simple variable name should be a lowercase letter. Later in the text we'll assign special significance to identifier that begin with a capital letter and to identifiers that use all capital letters.

Good Programming Practices

Multiple-word variable names can help make a program more readable. Separete the words with underscores as in total_commissions, or if you run the words together, begin each word fater the first with a capital letter as in totalCommissions. The latter style is preferred.

- Line 11 printf(“Enter first integer\n”); // prompt displays the literal “Enter first integer” and positions the curros the the beginning of the next line. This message is called a prompt because it tell the user to take a specific action.

```
1     // Addition program.
2     #include <stdio.h>
3
4     function main begins program execution
5     int main( void )
6     {
7     int integer1; // first number to be read from user
8     int integer2; // second number to be rad from user
9     int sum; // variable in which sum will be stored
10
11    printf( “Enter first integer\n” ); // prompt
12    scanf( “%d”, &integer1 ); // read an integer
13
14    printf( “Enter second integer\n” ); // prompt
15    scanf( “%d”, &integer2 ); // read an integer
16
17    sum = integer1 + integer2 // assign total to sum
18
19    printf( “Sum is %d\n”, sum  );
20    } // end function main
```

Fig 1.3 | Addition Program

- The next statement scanf(“%d”, &integer1); // read an integer uses scanf (the “f” stands for “formatted”) to obtain a value from the user. The function reads from the stadard input, which is usually the keyboard.

- This scanf has two arguments, “%d” and &integer1. The first, the format control string, indicates the type of data that should be entered by the user. The “%d” conversion specifier indicates that the data should be an integer (the letter d stands for decimal integer). The % in this context is treated by scanf as a speical character that begins a conversion specifier.

- The second argument of scanf begins with an ampersand (&)--called the address operator--followed by the variable name. The &, when combined with the variable name, tells scanf the location (or address) in memory at which the variable integer1 is stored.

Good Programming Practices

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

- When the computer executes the preceeding scanf, it waits for the user to enter a value for variable integer1. The user responds by typing an integer, the pressing the Enter key to send the number to the computer.

Another Simple C Program: Adding Two Integers--Assignment Statement

- The assignment statement in line 17 sum = integer1 + integer2; // assign total to sum calculates the total of variables integer1 and integer2 and assigns the result to variable sum using the assignment operator =.

- The = operator and the + operator are called binary operators because each has two operands. The + operator's two operands are integer1 and integer2. The = operators's two operands are sum and the value of the expression integer1 + integer2.

Good Programming Practices

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

Good Programming Practices

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

Another Simple C Program: Adding Two Integers--Printing with a Format Control String

- Line 19 printf(“Sum is %d\n”, sum); // print sum calls function printf to print the literal “Sum is %d\n” and sum. The first argument is the format control string. It contains some literal characters to be displayed, and it contains the conversion specifier %d indicating that an integer will be printed.

- The second argument specifies the value to be printed. Notice that the conversion specifier for an integer is the same in both printf and scanf--this is the case for most C data types.

Another Simple C Program: Adding Two Integers--Calculations in printf Statement

- Calculations can also be performed inside printf statements. We cuold have combined the previous two statements into the statement;

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

Good Programming Practices

Forgetting to precede a variable in a scanf statement with an ampersand when that variable should, in fact, be preceded by an ampersand results in an execution-time error. On many systems, this causes a “segmentation fault” or “access violation”. Such an error occurs when a user’s program attempts to access a part of the computer’s memory to which it does not have access privileges.

Arithmetic in C

Most C programs perform calculations using the C arithmetic operators given in Fig 1.4. The arithmetic operators are all binary operators. For example, the expression 3 + 7 contains the binary operator + and the operands 3 and 7.

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 x%y	x/y
Remainder	%	r mod s	r % s

Fig. 1.4 | Arithmetic operators

Introduction to C Programming

Arithmetic in C--Integer Division and the Remainer Operator

- Integer division yields an integer result. For example, the expression 7 / 4 evaluates to 1 and the expression 17 / 5 evaluates to 3.

- The remained operator is an integer operator that can be used only with integer operands. The expression x % y yields the remained after x is divided by y. Thus, 7 % 4 yields 3 and 17 % 5 yields 2.

Common Programming Error

An attempt to divide by zero is normally undefined on computer systems and generally results in a fata error, e.x., 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.

Introduction to C Programming

Arithmetic in C--Rules of Operator Precedence

- C applies the operators in arithmetic exprssions in a precise sequence determined by the following rules of operator precedence, which are generally the same as those in algebra:

  1. Operators in exprssions contained within pairs of parentheses are evaluated first. Parentheses are said to be at the highest level of precedence. In cases of nested, embedded parentheses such as ((a + b) + c) the operators in the innermost pair of parentheses are applied first.

  2. Multiplication, division and remainder operators are applied next. If an expression contains severel 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.

  3. 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.

  4. The assignment operator (=) is evaluated last.

- The rules of operator precedence specify the order C uses to evaluate expressions. When we say evaluation proceeds from left to right, we're referring to the associativity of the operators.

Introduction to C Programming

Arithmetic in C--Sample Algebraic and C Expressions

- Now let's consider severel expressions in light of the rules of operator precedence. Each example list ans algebraic expression and its C equivalent. The following expression calculate the arithmetic mean of five terms.

Algebra: $m = \frac{a + b + c + d + e}{5}$

C: `m = ( a + b + c + d + e ) / 5`

- If the parentheses are erronously omitted, we obtain a + b + c + d + e / 5, which evaluates incorrectly as

$$m = a + b + c + d + \frac{e}{5}$$

- The following expression contains remainder (%), multiplication, division, addition, subraction and assignment operations:

Algebra: `z = pr%q + w/x - y`

C:

```
z = p * r % q + w / x - y;
```

6 1 2 4 3 5

- As in algebra, its's acceptable to place unnecessray parentheses in an expression to make the expression clearer. These are called redundant parentheses.

Executable statements either perform actions (such as calculations or input or output of data) or make decisions.

This sections introduces a simple version of C's if statement that allows a program to make a decision based on the truth of falsity of a statement of fact called a condition. If condition is true, the statement in the body of the if statement is executed. If the condition is false, the body statement isn't executed. Whether the body statement is executed or not, after the if statement completes, execution proceeds with the next statement after the if statement.

Conditions in if statements are formed by using the equality operators and relations operators summarized in Fig. 1.5. The relational operators all have the same level of precedence and they associate left to right. The equality operators have a lower level of precedence than the relational operators and they also associate left to right.

Algebraic equality or relation 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 1.5 | Equality and relational operators

- Figure 1.6 uses six if statements to compare two numbers entered by the user. If the condition in any of these if statements is true, the printf statement associated with that if executes.

```
1   // Using if statements, relational operators, and equality operators.
2   #include <stdio.h>
3
4   // function main begins program execution
5   int main( void )
6   {
7     int num1; // first number to be read from user
8     int num2; // second number to be rad from user
9
10    printf( "Enter two integers, and I will tell you\n" );
11    printf( "the relationships they satisfy: " );
12
13    if ( num1 == num2 ){
14      printf( "%d is equal to %d\n", num1, num2 );
15    } // end if
16
17    if ( num1 != num2 ){
18      printf( "%d is not equal to %d\n", num1, num2 );
19    } // end if
20
21    if ( num1 < num2 ){
22      printf( "%d is less than %d\n", num1, num2 );
23    } // end if
24
```

```
25     if ( num1 > num2 ){
26        printf( "%d is greater than %d\n", num1, num2 );
27     } // end if
28
29     if ( num1 <= num2 ){
30        printf( "%d is less than or equal %d\n", num1, num2 );
31     } // end if
32
33     if ( num1 >= num2 ){
34        printf( "%d is greater than or equal %d\n", num1, num2 );
35     } // end if
36  } //end function main
```

Fig. 1.6 | Using if statements, relational operators, and equality operators.

- The program uses scanf to input two numbers. Each conversion specifier has a corresponding argument in which a value will be stored. The first %d converts a value to be stored in the variable num1, and the second %d converts a value to be stored in the variable num2.

Figure 1.7 lists from highest to lowest the precedence of the operators introduced in this chapter.

- Operatos are shown top to botton in decreasing order of precedence. The equals sign is also an operator. All these operators, with the exception of the assignment operator =, associate from left to right. The assignment operator = associates from right to left.

Operators	Associativity
()	left to right
* / %	left to right
+; -	left to right
< <= > >=	left to right
== !=	left to right
=	right to left

Fig. 1.7 | Precedence and associativity of the operators discussed so far.

Some of the words we've used in the C programs in this chapter--particularly int and if--are keywords or reserved words of the language. Figure 1.8 contains the C keywords. These words have special meaning to the C compiler, so you must be careful not to use these as identifiers such as variable names.

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

Fig. 1.8 | C's keywords

- We mentioned The CERT C Secure Coding Standard in the Preface and indicated that we would follow certain guidelines that will help you avoid programming practices that open systems to attacks.

- One such guideline is to avoid using printf with a single string argument. If you need to display a string that terminates with a newline, use puts function, which displays its string argument followed by a newline character. For example,

```
printf( “Welcome to C!\n” );
```

should be written as:

```
puts( “Welcome to C!” );
```

- If you need to display a string without a terminating newline character, use printf with two arguments, --a “%s” format control string and the string to display. The %s conversion specifier is for displaying a string. For example;

```
printf( “Welcome” );
```

should be written as:

```
printf( “%s”, “Welcome” );
```