

COP 3223: C Programming Spring 2009

Introduction To C - Part 2

Instructor : Dr. Mark Llewellyn
 markl@cs.ucf.edu
 HEC 236, 407-823-2790
<http://www.cs.ucf.edu/courses/cop3223/spr2009/section1>

School of Electrical Engineering and Computer Science
University of Central Florida



More Basic C Programming

- Continuing where we left off in the last section of notes, we'll continued to introduce some of the basic features of C in this section of notes.
- As we move on, note that the basics of C programming never change. We'll always use the same basic style of coding, and many of the statements we include in subsequent programs are the same we used in our very first program.
- However, pay attention because we will always be introducing new features of the language.



A Second C Program

Again the line numbers are for illustrative purposes – I simply turned on this option in the IDE. Do not type them in your code.

```
1 // sum of two integers (a second C program)
2 // This program adds two, user supplied, integers and prints their sum
3 // January 13, 2009   Written by: Mark Llewellyn
4
5 #include <stdio.h>
6
7 //main function
8 int main()
9 {
10     int integer1; //first integer to be entered by user
11     int integer2; //second integer to be entered by user
12     int sum;      //variable in which the sum will be stored
13
14     //write prompts to user and get numbers
15     printf("Enter first integer number\n");
16     scanf("%d", &integer1);
17     printf("Enter second integer number\n");
18     scanf("%d", &integer2);
19
20     sum = integer1 + integer2; //assign the total to sum
21     printf("The sum is %d\n", sum);
22     printf("\n\n");
23     system("PAUSE");
24
25     return 0;
26 } //end main function
```

```
C:\Courses\COP 3223 - C Progra...
Enter first integer number
23
Enter second integer number
45
The sum is 68

Press any key to continue . . . .
```



```
// sum of two integers (a second C program)
// This program adds two, user supplied, integers and prints their sum
// January 13, 2009   Written by: Mark Llewellyn

#include <stdio.h>

//main function
int main()
{
    int integer1; //first integer to be entered by user
    int integer2; //second integer to be entered by user
    int sum;      //variable in which the sum will be stored

    //write prompts to user and get numbers
    printf("Enter first integer number\n");
    scanf("%d", &integer1);
    printf("Enter second integer number\n");
    scanf("%d", &integer2);

    sum = integer1 + integer2; //assign the total to sum
    printf("The sum is %d\n", sum);
    printf("\n\n");
    system("PAUSE");

    return 0;
} //end main function
```

Text version of the program – maybe easier to read for input.



A Detailed Look At The Program

- The biggest difference between our first C program and this second C program is that this one requires the user to input two integer values for the program to add together. Our first program simply printed a line of text without requiring any input from the user.
- There are a number of different ways that input can be generated by the user to be entered into a program in C. We'll start with a very simple technique which uses a standard library function named `scanf` (which like `printf` is defined in the `stdio.h` file).

COMMON PROGRAMMING ERROR: It is important to note that standard library functions like `printf` and `scanf` are not part of the C programming language, rather they are part of a library of functions are available to C programs. This means, for example, that the compiler will not be able to detect spelling errors 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 – that is the linker’s job. When the linker runs, it locates the library functions and inserts the proper calls to these functions in the object code. Only now is the object program complete and ready for execution. If the function name is misspelled, the linker will spot the error, because it will be unable to match the name in the source program to a name of any known function in the libraries.



A Detailed Look At The Program

- We'll look more closely at formatted IO (input and output) later, but for now we'll just give a brief description of how this works.
- The `scanf` function is basically a pattern matching function that attempts to match up groups of input characters with conversion specifications (specified in the format control string). The generic format of a `scanf` call is:

```
scanf ( "format-control-string", input-variable-locations);
```

Example: `scanf("%d", &integer1);`

Format control string
contains conversion
specifiers

address of the
variable being
input

- The `scanf` function obtains its input from the standard input (which is typically the keyboard).



Some Commonly Used Conversion Specifiers

Conversion Specifier	Description
d	Read an optionally signed decimal integer. The corresponding argument is a pointer to an integer variable.
u	Read an unsigned decimal integer. The corresponding argument is pointer to an unsigned integer variable.
h or l	Place before any of the integer conversion specifiers to indicate that a <code>short</code> (h) or <code>long</code> (l) integer is to be input.
f	Read a floating-point value (e.g. 4.57). The corresponding argument is a pointer to a floating point variable.
c	Read a character. The corresponding argument is a pointer to a <code>char</code> variable.
s	Read a string. The corresponding argument is a pointer to an array of type <code>char</code> that is large enough to hold the string and a terminating null (<code>'\0'</code>) character – which is automatically added.



A Detailed Look At The Program

- Looking at line 16 of the program again:

```
scanf("%d", &integer1);
```

- Now you know that the "%d" string is the **format control string**, which in this case contains a single **conversion specifier**, indicating that the value to be input from the keyboard is a decimal integer.
 - The % sign in the format control string is treated by `scanf` (and `printf` as we will see later) as a special character that begins a conversion specifier. Each conversion specifier is preceded by a % sign.



A Detailed Look At The Program

- The second argument to the `scanf` call on line 16 begins with an `&`, which is called the **address operator** in C, and is followed by the variable name (`integer1`).
- The address operator, when combined with the variable name, tells the `scanf` function the location in memory at which the variable `integer1` is located.
- The computer then stores the value for `integer1` at that location .

COMMON PROGRAMMING ERROR:

While there are a few exceptions to this rule that we'll see later, in general, every variable in a `scanf` function call must be preceded by the `&` (address operator).



A Detailed Look At The Program

- Lines 15 and 17 of the program are simply prompts to the user indicating what they should do to interact with the executing program. Prompts are typically just `printf` statements with a message indicating what the user is to do to interact with the program.
- Line 20, `sum = integer1 + integer2;`, is called an **assignment statement**.
- Assignment statements are used to assign values to variables as the result of calculations made by the program.
- The **assignment operator** in C is the `=` sign.



A Detailed Look At The Program

- Lines 10, 11, and 12 of the program, are **variable definitions**, statements that define the **variables** that will be used by the program.
- Variables are simply logical names that the programmer assigns to hold values that the program will need to operate correctly.
- Variables correspond to locations in memory where the value of the variable is located.
- **In C, every variable must have a name and a type.**
- A variable name in C is any **valid** identifier. An identifier is a series of characters consisting of letters, digits and underscores that does not begin with a digit. Identifiers can be of any length, but only the first 31 characters are required to be recognized by the compiler.
- **C is case sensitive.** This means the `myNum` and `mynum` are different variable names.



Keywords In C

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

Cells highlighted in blue are keywords in the C99 standard only.



A Detailed Look At The Program

GOOD PROGRAMMING PRACTICE:

Multiple word variable names are typically constructed with the first word being lowercase and each additional word starting with an uppercase letter. For example, myNumberArray, or myName, rather than mynumberarray or myname.

GOOD PROGRAMMING PRACTICE:

While C allows variables to be defined using a “just in time” approach, meaning that as long as the variable is defined before it is references (used), then the definition is allowed. However, having variables declared throughout the program code makes reading/understanding the code more difficult. Therefore, a good convention to use is that in each function, all variables are declared before the first executable statement in the function. Further, separate the definitions from the executable statements by a single blank line.

GOOD PROGRAMMING PRACTICE:

Place spaces on both sides of binary operators in expressions to enhance readability. For example, use `sum = integer1 + integer2;` rather than `sum=integer1+integer2;`



A Detailed Look At The Program

- Lines 10, 11, and 12 of the program, are **variable definitions**, statements that define the **variables** that will be used by the program.
- Variables are simply logical names that the programmer assigns to hold values that the program will need to operate correctly.
- Variables correspond to locations in memory where the value of the variable is located.
- **In C, every variable must have a name and a type.**
- A variable name in C is any valid identifier. An identifier is a series of characters consisting of letters, digits and underscores that does not begin with a digit. Identifiers can be of any length, but only the first 31 characters are required to be recognized by the compiler.
- **C is case sensitive.** This means the `myNum` and `mynum` are different variable names.



A First Look At Simple Decisions

- Executable statements in C perform either **actions** (such as calculations or input/output of data) or **decisions** (we'll soon see several different types of decision statements).
- We'll start off looking at a very simple form of C's `if` statement. A simple if statement in C has the following form:

```
if ( condition )  
{  
    body of if statement  
}
```

- When a C program encounters an if statement, the validity of the condition is evaluated. If the condition (which is an expression that evaluates to either true (nonzero) or false(zero)) is true, the body of the if statement is executed. If the condition is false, the body of the if statement is not executed. In both cases the next statement to be executed is the next statement immediately following the if statement.



A First Look At Simple Decisions

- Let's construct a simple program that reads in three integer values from the keyboard and decides (using if statements) what the smallest and largest numbers are from the three values input.
- I'm going to construct this program in a not terribly efficient manner so that I can illustrate points about if statements, however, in the last practice problem in this set of notes you will be able to rewrite the program in a more efficient manner (if you think about the problem just a bit).
- Note, the program shown on the next page is not complete, in that I've cut away some of the details so that it will fit on one page, the full code is on our course webpage for you to use as you wish.




```

//define three integer variables to hold user input
int integer1, integer2, integer3;

//write prompts to user and get numbers
printf("Enter first integer number\n");
printf("Enter second integer number\n");
printf("Enter third integer number\n");

if (integer1 <= integer2 && integer1 <= integer3) {
    printf("The smallest value is: %d\n", integer1);
}

if (integer2 <= integer1 && integer2 <= integer3) {
    printf("The smallest value is: %d\n", integer2);
}

if (integer3 <= integer1 && integer3 <= integer2) {
    printf("The smallest value is: %d\n", integer3);
}

if (integer1 >= integer2 && integer1 >= integer3) {
    printf("The largest value is: %d\n", integer1);
}

if (integer2 >= integer1 && integer2 >= integer3) {
    printf("The largest value is: %d\n", integer2);
}

if (integer3 >= integer1 && integer3 >= integer2) {
    printf("The largest value is: %d\n", integer3);
}

```

This is the logical "AND" operator in C. The logical "OR" operator is || and the negation operator is !

The screenshot shows a window titled "C:\Courses\COP 3223 - C Program...". The terminal output is as follows:

```

Enter first integer number
45
Enter second integer number
23
Enter third integer number
77
The smallest value is: 23
The largest value is: 77

Press any key to continue . . . _

```



Truth Tables For AND, OR and NOT

A	B	A && B
T	T	T
T	F	F
F	T	F
F	F	F

AND

A	B	A B
T	T	T
T	F	T
F	T	T
F	F	F

OR

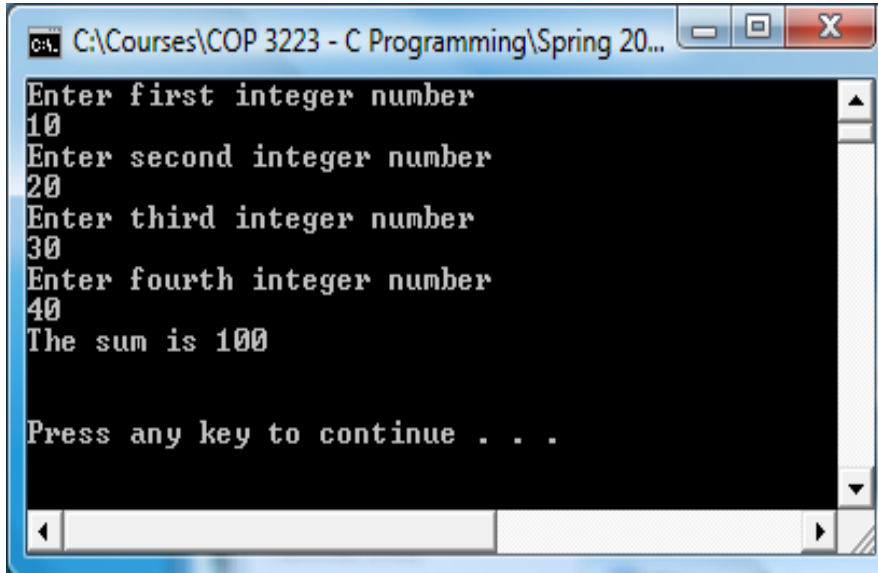
A	!A
T	F
F	T

NOT



Practice Problems

1. Modify the program on page 3 so that it accepts 4 integer values and produces the sum of the four numbers as output.



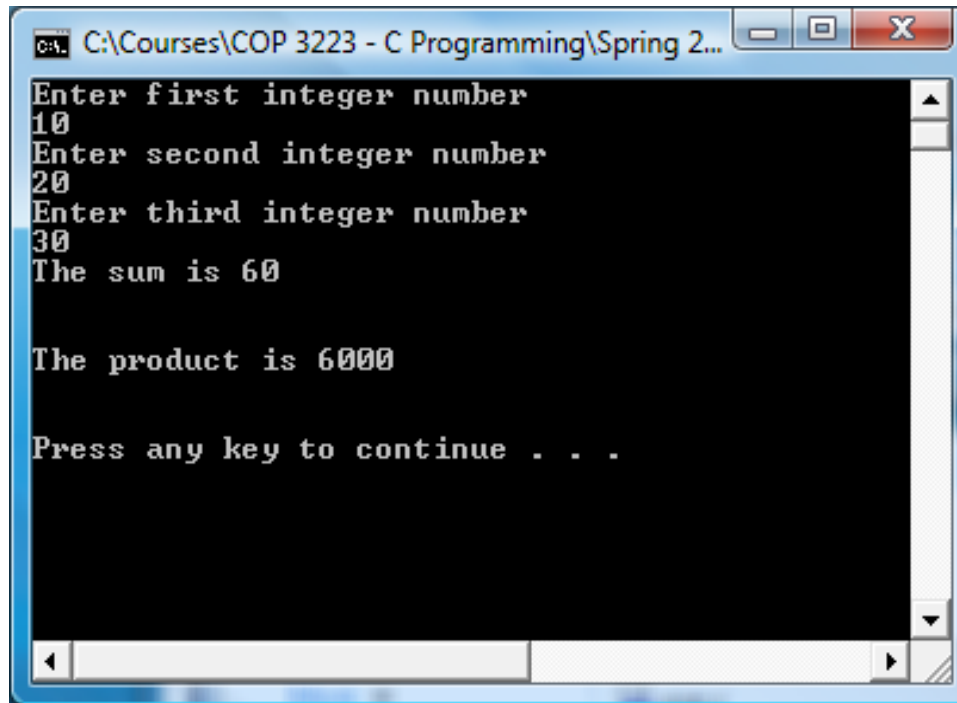
```
C:\Courses\COP 3223 - C Programming\Spring 20...
Enter first integer number
10
Enter second integer number
20
Enter third integer number
30
Enter fourth integer number
40
The sum is 100

Press any key to continue . . .
```



Practice Problems

2. Modify the program shown on page 3 so that it accepts three integer numbers and produces as output both the sum and product of the three numbers.



```
C:\Courses\COP 3223 - C Programming\Spring 2...
Enter first integer number
10
Enter second integer number
20
Enter third integer number
30
The sum is 60

The product is 6000

Press any key to continue . . .
```



Practice Problems

3. Modify the program shown on page 17, so that it does not require six if statements to determine the smallest and largest values from the input set. Modify it so that it still uses only if statements (i.e., don't use else statements or switch statements for those of you who may already know some more C than where we are currently at in the course.) Since I don't want you to use an else statement yet, you should be able to reduce the number of if statements from six to four. If somebody gets it in less than four if statements, let me know how you did it!

