INTRODUCTION TO C PROGRAMMING

Deitel 8th Edition, Chapter 2

TOPICS

A Simple C Program: Printing a Line of Text

Another Simple C Program: Adding Two Integers

Memory Concepts

Arithmetic in C

Decision Making: Equality and Relational Operators

A Simple C Program: Printing a Line of Text

Fig02_01.c – Parts of this C Program

Comments (lines 1, 2, 5, 9)

#include Preprocessor Directive (line 3)

Blank lines and white space (line 4, etc.)

A function called “main” (lines 6 – 9)

Curly braces (lines 7 & 9)

Output statement (line 8)

Semicolons (line 8)

Escape sequences

COMMENTS (LINES 1, 2, 5, 9)

Single-line comments begin with double forward slash **//**

Multi-line comments surrounded by **/*** and ***/**

Useful for long blocks of text

Tip!

In Dev-Cpp, highlight all the lines that should be commented out, and then press Ctrl + / on your keyboard.

This also works in reverse, to uncomment lines that have // at the beginning.

#INCLUDE PREPROCESSOR DIRECTIVE (LINE 3)

Lines beginning with **#** processed before compilation

These are called **preprocessor directives**

Include means to bring in other code, such as from C Standard Library

stdio.h is the Standard I/O Library header

- Needed for input and output

When the preprocessor executes, it will automatically bring this file into the program, so you can use the functions in it.

BLANK LINES AND WHITE SPACE (LINE 4, ETC.)

Improve readability

A FUNCTION CALLED "MAIN" (LINES 6 – 9)

Required for all C programs

Where execution begins

In this example, main starts at line 6, and ends at line 9.

This line:

int main (void)

Must appear in your source code file exactly as written.

All other executable statements will follow.

CURLY BRACES (LINES 7 & 9)

Required around code within a function, and other multi-line blocks of code

OUTPUT STATEMENT (LINE 8)

printf

String literal

- Characters within double-quotes

String literal within parentheses of a printf will appear on the screen

SEMICOLONS (LINE 8)

Required at the end of every executable line of code

ESCAPE SEQUENCES

Single character preceded by single backslash **** (called the escape character)

\n – means new line

FIG02_03.C, FIG02_04.C

Does same thing as 2.01, but with multiple printfs and newlines

Another Simple C Program: Adding Two Integers

FIG02_05.C – PARTS OF THIS PROGRAM

Reads two numbers as input from keyboard, adds them, and displays their sum

Variables and Variable Definitions (lines 8 & 9)

Prompting Messages (line 11)

The scanf Function and Formatted Inputs (line 12)

Prompting for and Inputting the Second Variable (lines 14 & 15)

Defining the sum Variable (line 17)

Assignment Statement (line 18)

Printing with a Format Control String (line 20)

VARIABLES & VARIABLE DEFINITIONS (LINES 8 & 9)

Also called **variable declarations**

In C, you must define (declare) variables before they are used

Must be defined with a name and a data type before using, or your code will not compile

More on this in a bit…

IDENTIFIERS AND CASE SENSITIVITY

Identifiers are the names of things – variables, functions, libraries, etc.

C is **case-sensitive**

Can include underscores, but no other special characters

- Don't use underscores at beginning of an identifier

Cannot begin with a digit

Use camel case: totalCommission

PROMPTING MESSAGES (LINE 11)

Displays text to screen

Positions cursor on next line (due to newline)

THE SCANF FUNCTION & FORMATTED INPUTS (LINE 12)

scanf obtains a value typed by the user via **stdin** (usually the keyboard)

This example has two parts within parens: “%d”, and &integer1

%d – **format control string**

- Indicates how the input data should be interpreted
- d means “decimal integer”
- Must use double-quotes and %

&integer1 – **location where the input will be stored**

- Address of the integer1 variable memory space
- Must use & for most inputs

THE SCANF FUNCTION & FORMATTED INPUTS (LINE 12), CONTINUED

Line 12 displays the cursor, waits for user to type and then press the Enter key

Then scanf reads what was typed and saves it in the variable noted in parentheses

Prompting for and Inputting the Second Variable (lines 14 & 15)

Same as lines 11 & 12, using the second variable

DEFINING THE SUM VARIABLE (LINE 17)

Defines it just before it is used

ASSIGNMENT STATEMENT (LINE 18)

Performs the operation on the right side of the equal sign, and saves that result in the variable on the left side of the equal sign

PRINTING WITH A FORMAT CONTROL STRING (LINE 20)

Has two arguments

String to be displayed, inside double-quotes

String may have **format control string** values embedded in it

Each fcs MUST have a value listed after the string

If more than one fcs within the string, the variables' values are substituted left to right

COMBINING VARIABLE DEFINITION & ASSIGNMENT STATEMENT

Called **initializing** the variable

Could combine lines 17 & 18:

```
int sum = integer1 + integer2;
```

Common practice, but be cautious when using this style

CALCULATIONS IN PRINTF STATEMENTS

Can use calculations instead of variable names

Also common, but be careful

MEMORY CONCEPTS

Memory Locations

Variable names correspond to **locations in memory**

Every variable has a **name, type, and value**

- And a location, which we'll discuss later in course

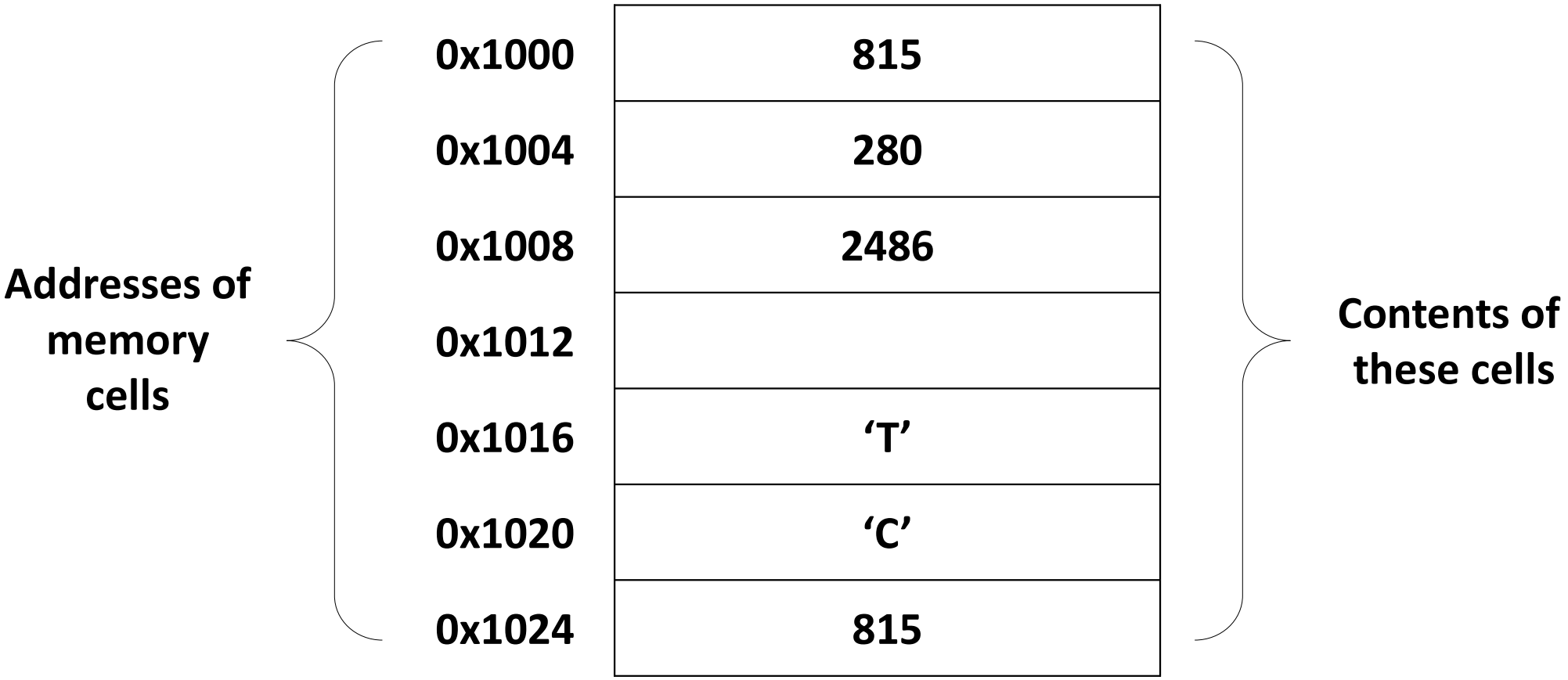
Anything **saved** in a memory cell overwrites any values that were there before

- Called "destructive"

Values in memory cells that are **used** (copied, read) are not destroyed

- Called "nondestructive"

VALUES IN MEMORY



Values in Memory – What if We Needed To Change a Value?

What if we needed to change **815** to **708**?

How does the OS know **which 815** to change?

0x1000	**815**
0x1004	**280**
0x1008	**2486**
0x1012	
0x1016	**'T'**
0x1020	**'C'**
0x1024	**815**

Values in Memory – We Need to Know Which value & Where it Is

We could tell the OS the **address of the cell** we want to change.

But we'd have to know the exact address first...

And knowing the exact address of a cell is the job of the OS, not the programmer.

0x1000	**815**
0x1004	**280**
0x1008	**2486**
0x1012	
0x1016	**'T'**
0x1020	**'C'**
0x1024	**815**

Values in Memory – So We Use Variable Names

So instead, within our program, we give these cells names.

When the program runs, the OS will reserve cells for use in our program and keep track of the names we give these cells.

These cells are called **variables**.

0x1000	**815**
0x1004	**280**
0x1008	**2486**
0x1012	
0x1016	**'T'**
0x1020	**'C'**
0x1024	**815**

Declaring a Variable

To **declare a variable** means to create a name for a memory cell

All data that exists while a program is running will reside in variables

All variables must be declared before you can use them in a program

Variable declarations must go within main*

Within a function, variable declarations go before other type of statements

*Or within other functions, as we'll see later

VARIABLE DECLARATIONS

To declare a variable, you must tell the compiler two things:

- **Type of data** that will be stored there
- **Name** to use for the cell

Can optionally give it an initial value

The OS will decide on the address

Semicolon at end of declaration is required

VARIABLE DECLARATIONS – EXAMPLES

```
int   area_code;

char   buildingLetter;

double   coffee_price;
```

There are three required parts to a variable declaration:

1. **Data type** – int, char, double in the above examples
2. **Variable name** – area_code, buildingLetter, coffee_price
3. **Semicolon - ;**

VARIABLE DECLARATIONS – EXAMPLES WITH INITIALIZATION

```
int   area_code = 815;

char   buildingLetter = 'T';

double   coffee_price = 3.94;
```

It is optional to also include an initial value.

This is called **initialization**.

ARITHMETIC IN C

Arithmetic in C Topics

Operators

Division in C

The remainder operator

Other tips

ARITHMETIC OPERATORS

Operation	Operator
Addition	+
Subtraction	-
Multiplication	*
Division	/
Remainder (mod)	%

DIVISION IN C

C performs **integer division**!

The result of division will use the same data type as the operands:

Integer result if the operands are integers

Double result if the operands are doubles

Division in C – Examples

Expression	Result in Math Class	Result in C Class
4 / 2	2	2
4.0 / 2.0	2.0	2.0
5 / 2	?	?
5.0 / 2.0	?	?

Division in C – Examples with Answers

Expression	Result in Math Class	Result in C Class
4 / 2	2	2
4.0 / 2.0	2.0	2.0
5 / 2	**2.5**	**2**
5.0 / 2.0	**2.5**	**2.5**

The Remainder Operator

Remainder operator %

Gives the remainder of integer division

- What's left when you can't divide anymore

So, 17 % 5 evaluates to 2

Remainder Operation in C

Expression	Result
4 % 2	0
5 % 2	1
9 % 3	0
8 % 3	2
10 % 10	0
15.0 % 3.0	Not defined

OTHER TIPS

Use parentheses for grouping & clarity

- Used the same way as in algebra

Remember the rules of operator precedence

- PEMDAS
- Assignment operator **=** is last
- See chart p. 51

Decision Making: Equality and Relational Operators

Equality and Relational Operators

Equality operators ==, !=

Relational operators <, <=, >, >=

Fig 2.13.c

- Note the parentheses around the **condition** in the if statement

Comparing Numbers (lines 17-19)

- Tests for equality

Use curly braces after an if