Programming in C

BY MARIA RAIDY

Info about C programming language

- **C** was developed by Dennis Ritchie at Bell Labs in the early 1970s for systems programming on the *PDP-11* computer
- Why is it called "**C**"?
  - It acquired its name from, an earlier language called *B*, which was written by Ken Thompson. B was a revision of a still earlier language, *bon*, which had been named after Thompson's wife, Bonnie.
- Most of the linux kernel is written in **C**
- **C** is seen as a low-level language (higher than assembly) since it can send hardware commands.

Setting up your computer

You need:

- A text editor: vi or gedit (to check if they are available, write their name on the command line and press ‘enter’)
- A compiler: GCC (GNU Compiler Collection) (check if it is installed)
- A C standard library: glibc (run the ‘locate glibc’ command)

The typical “Hello World!” program

Open your text editor and type the following lines of code:

```
#include <stdio.h>

main()
{
    printf("Hello World!\n");
    return 0;
}
```

Save your document and name it **hello.c**

! Do not name your program test or test.c because there is already a built-in command named test !

Explanation

```
#include <stdio.h> //a preprocessor directive copies the content of a header file
                   //keep an empty line as a matter of style and readability
main() //every C program has exactly 1 function called main that is the 1st to run
{      //opening curly braces

    //each statement ends with a semicolon (;)
    printf("Hello World!\n");   //printf() function and "\n" is the new line character
    return 0;           //the returned value is 0, we will use this later on
}      //closing curly braces
```

<u>Save your document and name it **hello.c**</u>

! Do not name your program test or test.c because there is already a built-in command naed test !

Compilation

1. Open the terminal and navigate to your directory.
2. Run the following command:
   - gcc -o hello hello.c OR gcc –o hello.out hello.c
   - The –o option specifies the name to be assigned to the compiled/executable program.
3. Run the ls command

Execution / Running

./hello OR ./hello.out

Program #2:

Asking the user for an input and printing it on the screen.

int

void

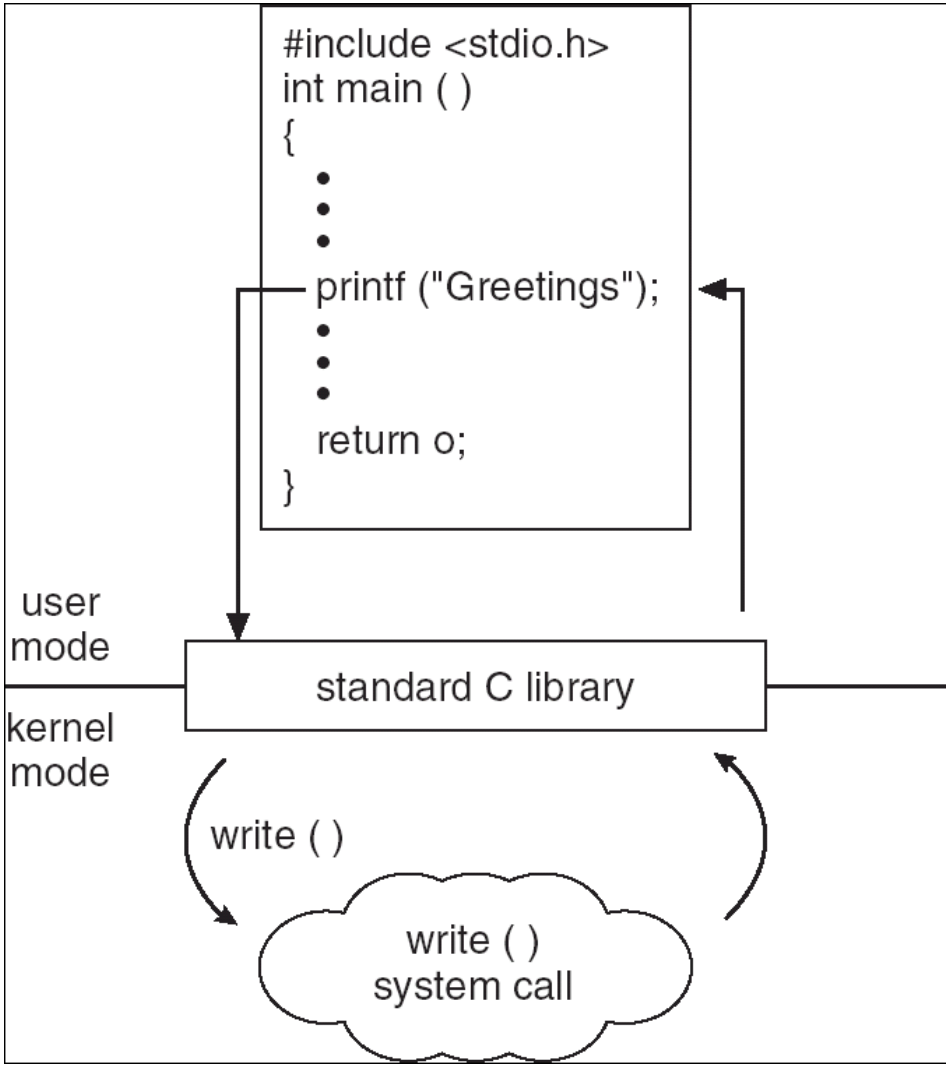
```
#include <stdio.h>

main()
{
    int count;
    puts("Please enter a number: ");
    scanf("%d", &count);          // %d is a placeholder
    printf("The number is %d \n", count);
    puts("The number is %d", count);
return 0;
}
```

Save, compile, run

What does every line do? What is the biggest number that a user can enter?

What happens if the user enters a character? A space in the middle of the number?


#include <stdio.h>
int main ( )
{
printf ("Greetings");
return o;
}
user mode
standard C library
kernel mode
write ( )
write ( )
system call

Basic data types and variables

- Integer: int %d
- Floating point number: float %f
- Characters: char %c
- Double: double
- Absence of a type: void

You can declare variables simultaneously if they are of the same type:

- int a, b, c;

Variables can be initialized (assigned an initial value) in their declaration. The initializer consists of an equal sign followed by a constant expression:

- type *var_name* = value;

Always initialize your variables!

Variable names

1. **Characters Allowed :**
   - Underscore(_)
   - Capital Letters (A – Z)
   - Small Letters (a – z)
   - Digits (0 – 9)
2. **Blanks & Commas** are not allowed
3. No Special Symbols other than **underscore(_) are allowed**
4. **First Character** should be **alphabet or Underscore**
5. Variable name must not be **Reserved Word**

32 keywords / reserved words

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

Write a program that:

1. Asks the user for his first name and waits for his input.
2. Then asks for his last name and waits for his input.
3. Then asks for his age and waits for his input.
4. <u>Display all the info as follow:</u>

   Full name: Maria Raidy

   Age: 22
5. <u>Save, compile, run</u>

Functions

- *A set of operations / a group of statements.*
- A function **declaration** tells the compiler about a function's name, return type, and parameters. [Always written **before** the main()]
- A function **definition** provides the actual body of the function.
- return_type function_name(parameter list)

```
{
        //body of the function
        return [value of return_type]
}
```

First function

Add to your last program a function called getGrades that asks the user to enter 2 of his grades and stores the values in 2 different variables called grade1 and grade2 (ex. grade1=75 and grade2= 80). It will then display the result as follows:

Grade of course 1: 75/100

Grade of course 2: 80/100

It doesn't return anything. What's its return type?

Now call this function from main: getGrades();

Save, compile, run.

Operators:
Arithmetic Operators

Operator	Description
+	Adds two operands.
–	Subtracts second operand from the first.
*	Multiplies both operands.
/	Divides numerator by de-numerator.
%	Modulus Operator and remainder of after an integer division.
++	Increment operator increases the integer value by one.
--	Decrement operator decreases the integer value by one.

Operators:
Relational Operators

Operator	Description
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.

Operators:
Logical Operators

Operator	Description
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.

Operators:
Assignment Operators

- =
- +=
- -=
- *=
- /=
- %=

Escape codes:

Similar to Java:

- \n
- \t
- \r
- \a
- \\
- \”

Guess!

Escape codes:

Similar to Java:

- \n new line
- \t tab
- \r carriage return
- \a alert
- \\ backslash
- \” double quote

```c
// Fig. 3.13: fig03_13.c
// Preincrementing and postincrementing.
#include <stdio.h>

// function main begins program execution
int main( void )
{
   int c; // define variable

   // demonstrate postincrement
   c = 5; // assign 5 to c
   printf( "%d\n", c ); // print 5
   printf( "%d\n", c++ ); // print 5 then postincrement
   printf( "%d\n\n", c ); // print 6

   // demonstrate preincrement
   c = 5; // assign 5 to c
   printf( "%d\n", c ); // print 5
   printf( "%d\n", ++c ); // preincrement then print 6
   printf( "%d\n", c ); // print 6
} // end function main
```

Fig. 3.13 | Preincrementing and postincrementing.

Function #2

Write a function called average that calculates the average of the 2 grades. It takes the 2 variables as parameters and returns a number.

If-else statement

```
if(boolean_expression)
 {
        /* statement(s) will execute if the boolean expression is true */
}
else
 {
        /* statement(s) will execute if the boolean expression is false */
 }
```

Switch case statement

```
switch(expression)  {
    case constant-expression :
      statement(s);
      break; // optional
    case constant-expression :
      statement(s);
      break; // optional
     /* you can have any number of case statements */
    default : //Optional
      statement(s);
 }
```

The ? : Operator

Exp1 ? Exp2 : Exp3;

- If exp1 is true, expr2 is evaluated and becomes the value of this expression.
- If exp1 is false, expr3 is evaluated and becomes the value of this expression.

While, do-while and for loops

while(condition)
{
statement(s);
}
	do
	{
	statement(s);
	} while(condition);

for (init; condition; increment)
{
statement(s);
}
	You can use one or more loops inside any other one.

Scope

Variables in C can be declared:

1. Inside a function or a block (ex loop) - ***local***
2. Outside all functions, before the main function - ***global***
3. In the definition of function parameters - ***formal***

Arrays

- It's a data structure that can store many values of the same type under one 'name'. Formally, it "stores a fixed-size sequential collection of elements". It can be multi-dimensional.
- You can declare a single variable called grades[2] and assign to it the values of grade1 and grade2.
- Syntax of declaration: type arrayName [size];
  type arrayName [rows] [columns];
- Initialization:

  Type arrayName[3]={value1, value2, value3};

  Type arrayName[]={value1, value2, value3};

  arrayName[index] = someValue;
- Update you program by declaring an array of grades instead of 2 variables.

Bonus function

Write a function called semesterAvg that:

1. Prompts the user to add more course grades in the form:
   - Enter grade (or -1 to end):
   - The user can keep on adding grades until he enters -1
2. Calculates the average of all the grades and returns it to the main function.

In the main function:

1. *Call this function*
2. *Display the result:*

   Your average this semester is XX.

Save, compile, run.

```
Enter grade, -1 to end: 75
Enter grade, -1 to end: 94
Enter grade, -1 to end: 97
Enter grade, -1 to end: 88
Enter grade, -1 to end: 70
Enter grade, -1 to end: 64
Enter grade, -1 to end: 83
Enter grade, -1 to end: 89
Enter grade, -1 to end: -1
Class average is 82.50
```

Little intro to pointers

- Every variable is a memory location.
- Every memory location has its address defined which can be accessed using ampersand (&) operator, which denotes an address in memory.
- Add a line of code to your program to display the address of one of your variables:
  - printf("Address of var1 variable: %x\n", &var1);
- A **pointer** is a variable whose value is the address of another variable.
- Declaration: type *pointer_name;
- Initialize as NULL pointer: type *pointer_name = NULL;
- Assigning a value: pointer_name = &var;

Pointers

- you can declare a pointer-to-pointer-to-type variable.
  - Ex: int **pptr;

- Ptr + 1 is the next memory location
- Ptr –1 is the previous memory location

- These arithmetic operators auto-adjust the address offset according to the type of the pointer.
- Ptr +1 = ptr + sizeof(int)
- Ptr +1 = ptr +sizeof(double)

Files (opening and closing)

- First, we need a pointer to keep track of the file we are accessing, in other words, it will be storing the address or the path/location of the file.
  - FILE *filePointer;
- Opening a file:
  - filePointer = fopen(filename, options);
    - r - open for reading
    - w - open for writing (file need not exist)
    - a - open for appending (file need not exist)
    - r+ - open for reading and writing, start at beginning
    - w+ - open for reading and writing (overwrite file)
    - a+ - open for reading and writing (append if file exists)
- Closing a file:
  - fclose (filePointer);

Files (writing)

```
#include <stdio.h>

main()
{
FILE *filePointer;
filePointer = fopen("/tmp/file1.txt", "w+");
fprintf(filePointer, "This is testing for fprintf.  \n");
fputs("This is testing for fputs.  \n", filePointer);
fclose(filePointer);
}
```

Files (reading)

```
#include <stdio.h>
main()
{
    FILE *filePointer;
    char str[255];      //str is a buffer
    filePointer = fopen("/tmp/file1.txt", "r");
    if ( filePointer == 0 ) { printf( "Could not open file\n" ); }
    else{
        Printf("0 : %s\n", fgetc(filePointer));

        fscanf ( filePointer, "%s", str );
        printf("1 : %s\n", str );

        fgets(str, 255, (FILE*)filePointer);
        printf("2: %s\n", str );

        fgets(file1, 255, (FILE*)filePointer);
        printf("3: %s\n", str );
        fclose(filePointer);
      }
}
```

```
while ( ( x = fgetc( filePointer ) ) != EOF )
{
    printf( "%c", x );
}
```

Practice Exercises

2.6 Identify and correct the errors in each of the following statements:

a)
```
printf( "The value is %d\n", &number );
```

b)
```
scanf( "%d%d", &number1, number2 );
```

c)
```
if ( c < 7 );{
   printf( "C is less than 7\n" );
}
```

d)
```
if ( c => 7 ) {
   printf( "C is greater than or equal to 7\n" );
}
```

Practice Exercises

2.7 Identify and correct the errors in each of the following statements. (*Note:* There may be more than one error per statement.)

a) `scanf( "d", value );`

b) `printf( "The product of %d and %d is %d"\n, x, y );`

c) `firstNumber + secondNumber = sumOfNumbers`

d)
```
if ( number => largest )
   largest == number;
```

e) `*/ Program to determine the largest of three integers /*`

f) `Scanf( "%d", anInteger );`

g) `printf( "Remainder of %d divided by %d is\n", x, y, x % y );`

h)
```
if ( x = y );
   printf( %d is equal to %d\n", x, y );
```

i) `print( "The sum is %d\n," x + y );`

j) `Printf( "The value you entered is: %d\n, &value );`

3.8 Identify and correct the errors in each of the following:

a)
```
while ( c <= 5 ) {
   product *= c;
   ++c;
```

b) `scanf( "%.4f", &value );`

c)
```
if ( gender == 1 )
   puts( "Woman" );
else;
   puts( "Man" );
```

Practice Exercises

3.12 What does the following program print?

```c
#include <stdio.h>

int main( void )
{
   unsigned int x = 1, total = 0, y;

   while ( x <= 10 ) {
      y = x * x;
      printf( "%d\n", y );
      total += y;
      ++x;
   } // end while

   printf( "Total is %d\n", total );
} // end main
```

3.23 ***(Find the Largest Number)*** The process of finding the largest number (i.e., the maximum of a group of numbers) is used frequently in computer applications. For example, a program that determines the winner of a sales contest would input the number of units sold by each salesperson. The salesperson who sells the most units wins the contest. Write a pseudocode program and then a program that inputs a series of 10 non-negative numbers and determines and prints the largest of the numbers. *Hint:* Your program should use three variables as follows:

`counter:`	A counter to count to 10 (i.e., to keep track of how many numbers have been input and to determine when all 10 numbers have been processed)
`number:`	The current number input to the program
`largest:`	The largest number found so far

Practice Exercises

3.34 ***(Palindrome Tester)*** A palindrome is a number or a text phrase that reads the same backward as forward. For example, each of the following five-digit integers is a palindrome: 12321, 55555, 45554 and 11611. Write a program that reads in a five-digit integer and determines whether or not it's a palindrome. [*Hint*: Use the division and remainder operators to separate the number into its individual digits.]

2.21 ***(Shapes with Asterisks)*** Write a program that prints the following shapes with asterisks.

```
*********          ***            *             *
*       *        *     *         ***           * *
*       *       *       *       *****         *   *
*       *       *       *         *           *     *
*       *       *       *         *           *       *
*       *       *       *         *           *     *
*       *       *       *         *           *   *
*       *        *     *          *           * *
*********          ***            *             *
```

More info on:

- https://www.tutorialspoint.com/cprogramming/c_file_io.htm
- https://www.cprogramming.com/tutorial/cfileio.html
- http://www.c4learn.com/c-programming/c-keywords/
- http://www.linfo.org/c.html
- “Learning C” by Neil GRAHAM, McGraw-Hill International Editions
- “C how to program” by Deitel and Deitel