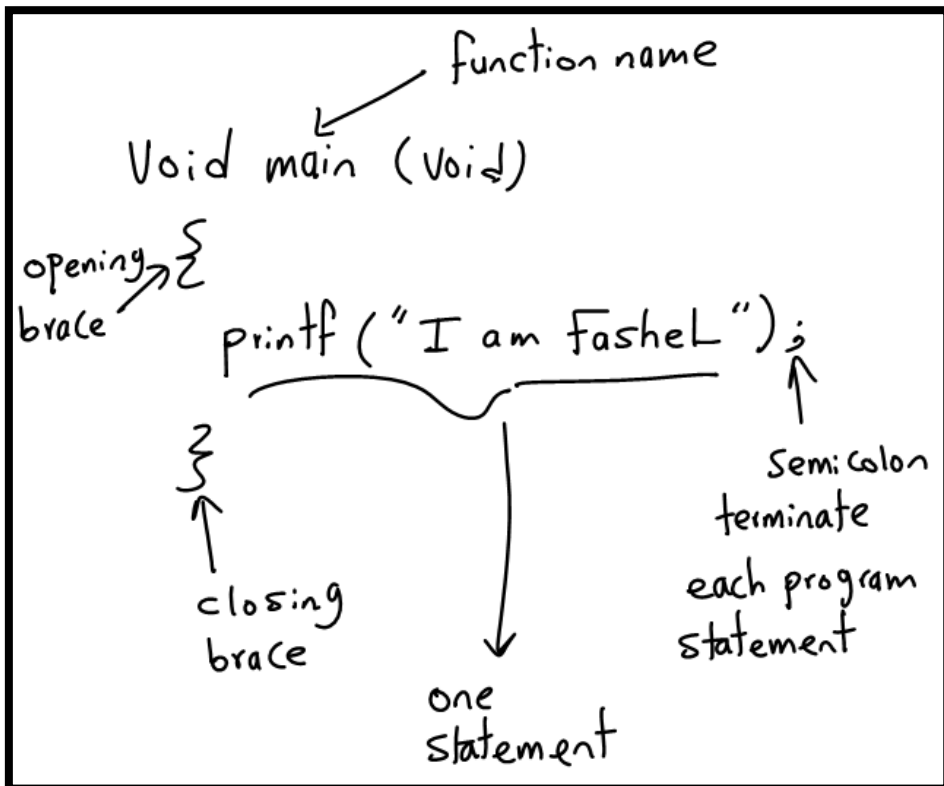


Chapter 03

Writing a very simple C Program

- Learning C, as is true with any language, is largely a matter of practice. The more you look at C programs, the less complex they appear, until at some point you wonder why you ever thought they looked complicated



- All C programs are divided into units called **Functions**.
- A **function** is similar to **subroutine** in BASIC or a **function** in PASCAL
- Note that **main** is a function
- Every C program consists of one or more functions — this program has only one

- No matter how many functions there are in a C program, **main()** is the one to which control is passed from the operating system when the program is run — it is the first function executed
- The word "**Void**" preceding "main" specifies that the function main() will not return a value. The second "void," in parentheses, specifies that the function takes no arguments
- The **whitespace characters** (space, tab, newline) are invisible to the compiler

LAB 01

Write and execute the following file

```
Void main (void){printf("I am Fashel");}
```

- **Indentation** of blocks of code enclosed in braces is an important aspect of making C programs readable
- The program line `printf("I am Fashel");` causes the phrase in quotes to be printed on the screen
- The word **printf** is actually a **function name**
- Since **printf** is a function, it is followed by **parentheses**

- The phrase is a function **argument** — information passed from `main( )` to the function `printf( )`
- The parentheses following the name just to make it clear that we are talking about a function and not a variable or something else
- "I am Fashel" is a string constant — string constant is surrounded by quotes; this is how the compiler recognizes a string
- Notice that we have used the function `printf` in our program, but there is no code for this function in our program. Where is `printf` stored? It is in `stdio.h` file — standard input output header file
 - When compiling the program, the compiler realizes that `printf` is not a function included in your own program source file, so it leaves a message to that effect for the linker
- C distinguishes between uppercase and lowercase

- Exploring the **printf** function

Printf

```
Void main(void)
{
    Printf("this is the number two:%d",2)
}
```

- Can you guess what message will be printed when you run it

this is the number two:2

- The function `Printf()` can be given more than one argument
- These two arguments are separated by a comma
 - "this is the number two:%d"
 - 2
- The `Printf()` function takes the value on the right of the comma and plugs it into the string on the left – in place of **format specifier**
- Why not simply put the number 2 into the original string?
 - `Printf("this is the number two:%d",2)`
 - It wouldn't make any difference in what is printed out, since 2 is a constant

- **Variables** can also be used in the `Printf( )` function giving it the capability of changing what it prints while the program is running

Format specifier

```
Void main(void)
{
    Printf("%s is %d million miles\n from the
           sun", "venus", 67);
}
```

- The output will be
Venus is 67 million miles
From the sun
- The `Printf( )` function has replaced the `%s` symbol with the string "Venus" and the `%d` symbol with the number 67

A handwritten diagram of the `printf` function call: `printf ("%s is %d million mile\n from the sun.", "Venus", 67);`. The string format specifiers `%s` and `%d` in the format string are each circled in blue and labeled with a circled '1' and '2' respectively. Red arrows point from these labels to the corresponding arguments in the argument list: `"Venus"` (labeled with a circled '1') and `67` (labeled with a circled '2').

LAB 02

Write and execute the following file

```
main( )
{
    printf("Hello, ");
    printf("world");
    printf("\n");
}
```

- **'\n'** means "**newline**" and stands for a single character that, when inserted in a string, has the effect of a carriage return and linefeed – **escape sequence**
- The newline character is stored in memory as a single character – a linefeed – but it has the effect of both a carriage return and linefeed

LAB 03

Write and execute the following file

```
main( )
{
    printf("the letter %c is", 'j');
    printf("pronounced %s.", "jay");
}
```

- Notice that 'j' is surrounded by single quotes — apostrophes
- While "jay" is surrounded by double quotes — this is how the compiler tells the difference between a character and a string

- If you want a newline character, you must explicitly insert it
- An **escape sequence** like `\n` provides a general and extensible mechanism for representing hard-to-type or invisible characters
- Among the others that C provides are `\t` for tab, `\b` for backspace, `\"` for the double quote, `\\` for the backslash itself

Quiz

What is the output of the following program?

```
main( )
{
    printf("%s\n%s\n%s", "one", "two", "three");
}
```

- **Variables**
 - A variable is a space in the computer's memory set aside for a certain kind of data and given a name for easy reference
 - What different kinds of variables does the language recognize? How is this data stored? How do you tell the computer what sort of data you want to store?

Variable

```
main( )  
{  
    int num  
    num = 2;  
    printf("This is the number two: %d",num);  
}
```

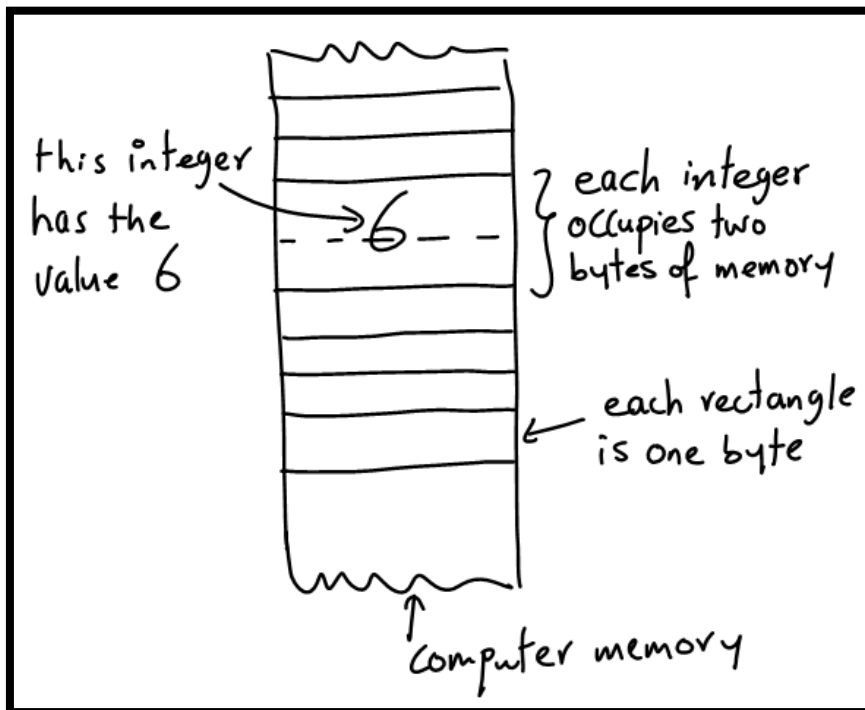
It creates a variable num, assigns it the value 2, and then prints out the value contained in the variable

- A variable declared, it is given a name and a type (type int, which we will examine soon)
 - int num;
- the variable is assigned a value
 - num =2
- in the third statement of the program, the variable name, num is used as one of the arguments of the printf() statement, replacing the constant 2 used.
- **Variable definitions**

- The definition consists of the type name, int, followed by the name of the variable, num
- In a C program all variables must be defined

- If you have more than one variable of the same type, you can define them all with one type name, separating the variable names with commas

```
int id, count, index
```
- When you define a variable, the compiler sets aside an appropriate amount of memory to store that variable — in the present case we've specified an integer variable, so the compiler will set aside two bytes of memory
- Actually, the amount of memory used for an integer (and other variable types) is not defined as part of the C language — it is dependent on the particular computer system and compiler being used



- Variable definition specifies a name and type of variable and also set aside memory space for the variable
- Variable declaration specifies variable name and data type but does not set aside any memory for the variable
- A character variable is one-byte space in memory in which the character constants such as 'a' or 'X' can be stored — `Char ch1, ch2;`
- Floating point numbers are used to represent values that are measured — **float – double**

- **Format specifier**

- | |
|---|
| ▪ In general, we want to match the format specifier to the type of variable you're printing |
| %c single character |
| %s string |
| %d signed decimal integer |
| %f floating point |
| %e floating point "exponential notation" |
| %g floating point (%f or %e, whichever is shorter) |
| %u unsigned decimal integer |
| %x unsigned hexadecimal integer (uses "abcdef") |
| %o unsigned octal integer |

- The format specifier in `printf( )` determines the interpretation of a variable's type, the width of the field, the number of decimal places printed, and the justification

▪ **Printing Graphics Characters**

- As you probably know, every character (letters, digits, punctuation, and so forth) is represented in the computer by a number
- True ASCII codes run from 0 to 127 (decimal). These cover the upper and lowercase letters, digits 0 to 9, punctuation, and control characters such as linefeed and back-space
- PC computers use an additional 128 characters, with codes running from 128 to 255. These additional characters consist of foreign-language symbols and graphics character
- Graphics characters are printed using their hex code in an escape sequence, such as `'\xB0'`

Graphics

```
main( )
{
    printf("\xC9\xCD\xBB\n");
    printf("\xC8\xCD\xBC\n");
}
```

▪ **Scanf() Function**

- **Scanf()** is an important input function

```
#include <stdio.h>

main()
{
    float years,days;
    printf("Please type your age in years:");
    scanf("%f",&years);
    days = years*365;
    printf("You are %f days old",days);
}
```

- Address operator represented by the ampersand (&)
- Simply note that `scanf( )` requires the use of an ampersand before the variable name

```
Please type your age in years:41
You are 14965.000000 days old
```

- The `scanf( )` function can accept input to several variables at once

```
scanf("%f %f",&x,&y);
```

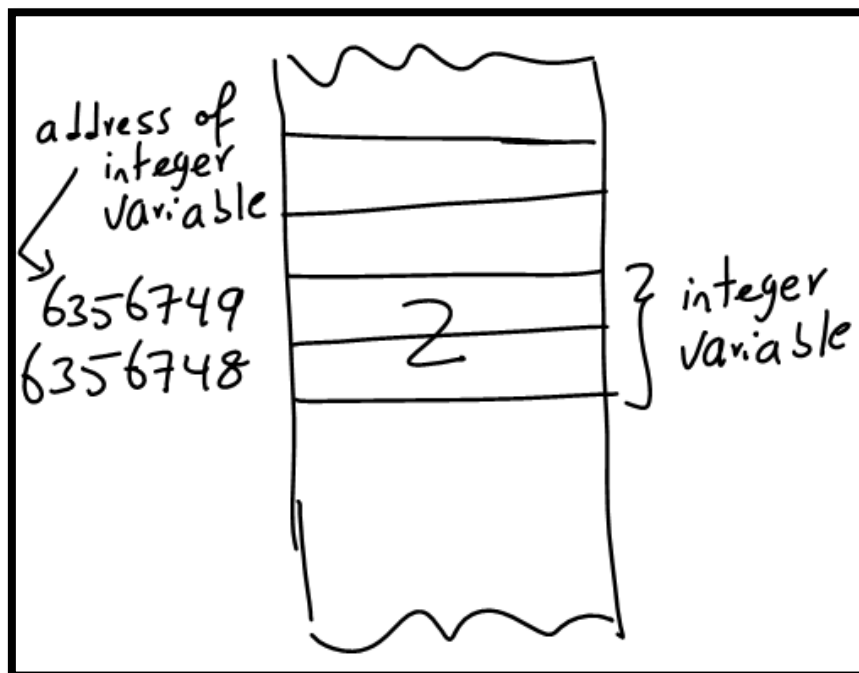
- The C compiler requires the arguments to `scanf( )` to be the address of the variables rather than the variables themselves
- The memory of your computer is divided into bytes, and these bytes are numbered, from 0 to the upper limit of

your memory. These numbers are called the "addresses" of the bytes.

- Every variable occupies a certain location in memory, and its address is that of the first byte it occupies

```
#include <stdio.h>
main()
{
    int num;
    num =2;
    printf("Value    =    %d\t    Address    =
%d", num, &num);
}
```

Value = 2 Address = 6356748



- In any case, knowing where a variable is stored will turn out to be very important in C programming, as will be covered under the **Pointer** title
- In the meantime, all you need to precede variable names with the ampersand (except in the case of strings, which we're coming to soon)

- **The `getche( )` function**

- The `scanf( )` function has one weak point — you need to type a [Return] before the function will digest what we have typed
- But we often want a function that will read a single character the instant it is typed without waiting for [Return].
- For instance, in a game we might want a space-ship to move each time we pressed one of the cursor control (arrow) keys — we can use `getche( )` for this purpose
- `getche( )` → get character with echo
- `getch( )` → get character without echo
- `getchar( )` → get character — it does not pass the character typed by the user to the program until the user hits [Return]

getche()

```
main( )
{
    Char ch
    printf("Type any character:");
    ch = getche()
    printf("\nThe character you typed was
    %c.",ch);
}
```

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  main()
5  {
6      char ch;
7      printf("Type any character:");
8      ch = getche();
9      printf("\nThe character you typed was %c.",ch);
10 }
11
```

Type any character:r
The character you typed was r.

- there is a downside to using `getche( )`; if you make a mistake, you can't backspace to correct it, since as soon as you type a character, it is delivered to your program
- we'll see how useful **`getche( )`** can be, when we learn such skills as how to count characters in phrases typed in by user.

▪ Operators

- Operators are words or symbols that cause a program to do something to variables
- For instance, the arithmetic operators (+) and (-) cause a program to add or subtract two numbers
- There are many different kinds of operators — mention the most common
 - Arithmetic
 - Relational
 - Increment/decrement
 - Assignment

▪ Arithmetic operators

+	addition
-	subtraction
*	multiplication
/	division
%	remainder

- Multiplication and division are normally carried out before addition and subtraction
- That fact that (*) and (/) are evaluated before (+) and (-) is an example of **precedence**

- We say that (*) and (/) have a higher precedence than (+) and (-)
- The remainder operator (modulo operator) may be unfamiliar to you — it is used to find the remainder when one number is divided by another
`Answer = 13 % 5`
- The variable answer will be assigned the value 3, since that is the remainder when 13 is divided by 5

Fahrenheit to centigrade

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    float ftemp, ctemp;
    printf("Type temperature in degree
fahrenheit: ");
    scanf("%f", &ftemp);
    ctemp = (ftemp-32)*5/9;
    printf("Temperature in degrees centigrade is
%f", ctemp);
}
```

```
Type temperature in degree fahrenheit: 70
Temperature in degrees centigrade is 21.111111
```

- It is also possible to include expressions involving arithmetic operators (and other kinds of operators as well) directly into `printf( )` and other kinds of statements

Fahrenheit to centigrade

```
#include <stdio.h>
main()
{
    int num;
    printf("number plus 4 is %d",num+4);
}
```

- An entire expression can be used almost anywhere a variable can be used

■ Arithmetic Assignment Operator

- If you compare a C program with a program with the same purpose written in another language, you may well find that the C source file is shorter — C has several operators that can compress often-used programming statements

Total = total +number	➔	total+=number
Total = total -number	➔	total-=number
Total = total *number	➔	total*=number
Total = total /number	➔	total/=number
Total = total %number	➔	total%=number

■ The Increment operator

- The increment operator ++ adds 1 to its operand,
- while the decrement operator -- subtracts 1
- ++ and – may be used either as prefix operators (before the variable, as in ++n or postfix (after the variable n++) — in both cases, the effect is to increment n
- But the expression ++n increments n before its value is used
- While n++ increments n after its value has been used.
- If n = 5, then
 - X = n++;
 - Sets x to 5
- X = ++n
 - Sets x to 6
- In both cases n becomes 6

Increment/Decrement

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    int num= 10;
    printf("number = %d", num++);
    printf("\nnumber = %d", num);
    num = 10;
```

```
printf("\nnumber = %d", ++num);  
printf("\nnumber = %d", num);  
}
```

```
number = 10  
number = 11  
number = 11  
number = 11
```

■ Relational Operator

- it will be used while dealing with loops and decisions
- is a certain variable greater than 100? — if so stop looping
- is the character the user just typed equal to a space? If so, increment the count of the number of words
- < less than
- > greater than
- <= less than or equal to
- >= greater than or equal to
- == equal to
- != not equal to

Relational operator

```
#include <stdio.h>  
#include <stdlib.h>  
main()  
{  
    int age;
```

Chapter 03

Writing a very simple C Program

```
age = 15;
printf("Is age less than 21? %d\n", age < 21);

age = 30;
printf("Is age less than 21? %d\n", age < 21);
}
```

```
Is age less than 21? 1
Is age less than 21? 0
```

- In C, true is represented by the integer 1, and false values are represented by the integer 0
- Note that the relational operator "equal to" is represented by two equal signs — a common mistake is to use a single equal sign as a relational operator.
- Any nonzero integer is recognized by compiler as true value
- Arithmetic operators have a higher precedence — that is, are evaluated before — relational operators

Precedence

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    Printf("Answer is %d", 2+1<4);
    Printf("\nAnswer is %d", 2+(1<4));
}
```

```
Answer is 1  
Answer is 3
```

■ Comments

- It is helpful to be able to put comments into the source code file that can read by humans but are invisible to the compiler
- A comment begins with the two-character symbol slash-asterisk (/*) and ends with an asterisk-slash (*/)
- Comments should be added anyplace there's possibility of confusion

Comments

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  main()
5  {
6      float years,days;           /* initialize variab
7      printf("Please type your age in years: "); /* print prompt*/
8      scanf("%f",&years);         /* get age in years
9      days = years*365 ;          /* calculate age in
10     printf("You are %f days old",days);      /* print result*/
11
12
13 }
14
```

■ Sheet

1. Write a C program that reads in the radius of a circle, calculates its area and then writes the calculated result.
- 2.

```
#include <stdio.h>

int main()
{
    printf("Braces come in pairs!");
    printf("Comments come in pairs!");
    printf("All statements end with a semicolon!");
    printf("Spaces are optional!");
    printf("Must have a main function!");
    printf("C is done mostly in lowercase.\n");
    printf("    It's a case-sensitive language.");
    return(0);
}
```

Type the preceding source code into your editor. Save the code to disk as RULES.C. Compile and run

3. The height of an object dropped from a starting height of 100 meters is given by the equation $h = 100 - 4.9 * t^2$ where t is the time in seconds. Write a C program that asks the user for a time less than 20 seconds and displays the height of the object at that time using this formula. The program output should look similar to:

Enter the time:2

The object is at 80.4 meters

4. The distance to school is 32 Kilometres. Write a C program that asks the user for an average speed in Kilometres per hour,

and then displays the time in minutes required for the trip at this speed. The program output should look similar to:

```
Enter the average speed: 88
```

```
The trip to school will take 21.8 minutes
```

5. Einstein's famous formula, $E = MC^2$ gives the amount of energy released by the complete conversion of matter of mass M into Energy E . if M represents the mass in kilograms and C represents the speed of light in meters per second ($3.0 \times 10^8 \text{ m/s}$), then the result is in the energy unit Joules. It takes **360000** Joules to light a **100-watt** light bulb for an hour. Write a **C** program that calculates the energy and the number of light bulbs that could be powered by a given mass of matter. The program output should look similar to:

```
Enter the mass in kg: 0.01
```

```
This mass could produce 9e+14 Joules of  
energy.
```

```
It could power 2.5e+09 100-watt light  
bulbs for an hour
```

6. Write a C program that allows the user to enter two integers, and then displays the results of integer (/) and modulus (%) division in either order. Program output should look similar to:

```
Enter first integer: 13
```

```
Enter second integer: 4
```

$13/4 = 3$

$13\%4 = 1$

$4/13 = 0$

$4\%13 = 4$

7. Write a C program that allows the user to enter a time in minutes and then displays the time in hours: minutes format.

The program output should look similar to:

Enter the number of minutes:327

This is 5:27

8. Write a program that displays the coins necessary to make the change entered by the user. The change can be made up of 1,5,10,50 pounds. The program output should look similar to:

Enter the amount of change in Egyptian pound:79

Change:

No. of 50:1

No. of 10:2

No. of 5 :1

No. of 1 :4

9. Write a program that accepts a two-digit number and displays the digits each on single line. The program should look similar to:

Enter a two-digit number: 27

The first digit is 2

The second digit is 7

10. Modify the program to report the sum of the digits of a three-digit number:

Enter a three-digit number: 274

The first digit is 2

The second digit is 7

The third digit is 4

The sum of the digits is 13

11. Write a program that calculates the number of hours of your life that you have spent sleeping. Assume that you sleep 8 hours each night. To simplify the problem, assume that there are always 30 days in each month, and 365 days in each year. The program output should look similar to:

Enter your birthdate (using numbers for months):

Month: 9

Day: 27

Year: 1984

Enter today's date (using numbers for months):

Month: 10

Day: 26

Year: 2002

You have been alive for 6599 days

You have slept 52792 hours

12. A fast food restaurant charges 20.5 LE for burgers, 10 LE for fries, and 5.5 for sodas. Write a C program that allows an employee to enter an order (the number of burgers, fries, sodas), and then display a total, the tax (at 6%), and then the final cost. The program should look similar to:

Enter the number of burgers: 3

Enter the number of fries: 2

Enter the number of sodas: 5

The total before tax is: 11.20

The tax is 0.67

The grand total is 11.87