Before we can begin to write programs in C, it would be interesting to find out what really is C, how it came into existence and how does it compare with other programming languages. In this chapter, we would briefly outline these issues.

Four important aspects of any language are—the way it stores data, the way it operates upon this data, how it accomplishes input and output, and how it lets you control the sequence of execution of instructions in a program. We would discuss the first three of these building blocks in this chapter.

What is C?

C is a programming language developed at AT & T's Bell Laboratories of USA in 1972 by Dennis Ritchie. C became popular because it is simple and easy to use. An opinion that is often heard today is—"C has been already superseded by languages like C++, C# and Java, so why bother to learn C today". I seriously beg to differ with this opinion. There are several reasons for this. These are as follows:

(a) C++, C# or Java make use of a principle called Object Oriented Programming (OOP) to organize programs which offers many advantages. While using OOP, you need basic programming skills. So, it makes more sense to first learn C and then migrate to C++, C# or Java. Though this two-step learning process may take more time, but at the end of it, you will definitely find it worth the trouble.

(b) Major parts of popular operating systems like Windows, UNIX, Linux and Android are written in C. Moreover, if one is to extend the operating system to work with new devices, one needs to write device driver programs. These programs are written exclusively in C.

(c) Common consumer devices like microwave ovens, washing machines and digital cameras are getting smarter by the day. This smartness comes from a microprocessor, an operating system and a program embedded in these devices. These programs have to run fast and work in limited amount of memory. C is the language of choice while building such operating systems and programs.

(d) You must have seen several computer games where the user navigates some object, like say a spaceship and fires bullets at invaders. The essence of all such games is speed. To match this expectation of speed, the game has to react fast to the user inputs. The popular gaming frameworks (like DirectX) that are used for creating such games are written in C.

I hope that these are very convincing reasons why you should adopt C as the first step in your quest for learning programming.

Which C are we Learning?

The official description of the C programming language was published by Brian Kernighan and Dennis Ritchie in 1978. It is commonly referred as K&R.

In 1983, the American National Standards Institute formed a committee, X3J11, to establish a standard specification of C. The ANSI standard was completed in 1989 and ratified as ANSI X3.159-1989 "Programming Language C." This version of the language is often referred to as "ANSI C". It is sometimes also known as C89.

In 1995, the ISO published an extension to ANSI C which is often referred as ISO C. In March 2000, ANSI adopted ISO C. This standard is commonly referred to as C99. All programs in this book are based on C99 standard.

Getting Started with C

There is a close analogy between learning English language and learning C language. This is illustrated in the Figure 1.1.

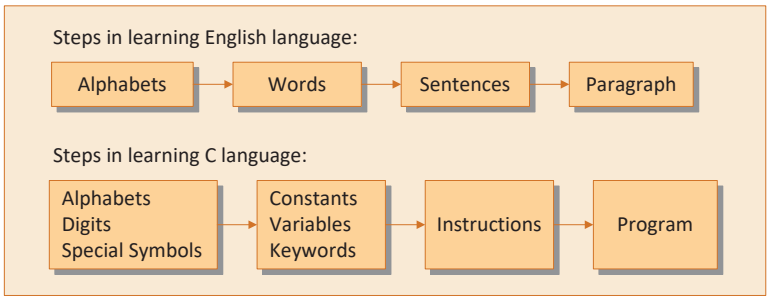


Figure 1.1 Steps in learning a language.

As with English, C language too has a set of rules that one must follow while writing programs in it.

Alphabets, Digits and Special Symbols

Figure 1.2 shows the valid alphabets, numbers and special symbols allowed in C.

Alphabets	A, B, ..., Y, Z a, b, ..., y, z
Digits	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Special symbols	~ ‘ ! @ # % ^ & * ( ) _ - + = | \ { } [ ] : ; " ' < > , . ? / $

Figure 1.2 Alphabets, digits, special symbols used in C.

Constants, Variables and Keywords

The alphabets, digits and special symbols when properly combined form constants, variables and keywords. A constant is an entity that doesn't change, whereas, a variable is an entity that may change. A keyword is a word that carries special meaning. In programming languages, constants are often called literals, whereas variables are called identifiers. Let us now see what different types of constants and variables exist in C.

Types of C Constants

Constants in C can be divided into two major categories:

(a) Primary Constants – Integer, Real, Character
(b) Secondary Constants – Pointer, Array, String, Structure, Union, Enum

At this stage, we would restrict our discussion to only Primary constants, namely, Integer, Real and Character constants. Following Rules have been laid down for constructing these different types of constants:

Rules for Constructing Integer Constants

(a) An integer constant must contain at least one digit.
(b) It must not contain a decimal point.
(c) Its value can be zero, positive or negative. If no sign precedes an integer constant, it is assumed to be positive.
(d) Commas or blanks are not allowed within an integer constant.
(e) The allowable range for integer constants is -2147483648 to +2147483647.

Ex.: 426 +782 -8000 -7605

Truly speaking, the range of an Integer constant depends upon the compiler. For compilers like Visual Studio, GCC, it is -2147483648 to +2147483647, whereas for compilers like Turbo C or Turbo C++, the range is -32768 to +32767.

Rules for Constructing Real Constants

Real constants are often called Floating Point constants. Real constants could be written in two forms—Fractional form and Exponential form. Following rules must be observed while constructing real constants expressed in fractional form:

(a) A real constant must contain at least one digit.

(b) It must contain a decimal point.

(c) It can be zero, positive or negative. Default sign is positive.

(d) Commas or blanks are not allowed within a real constant.

Ex.: +325.34 426.0 -32.76 -48.5792

The exponential form is usually used if the value of the constant is either too small or too large. It, however, doesn't restrict us from using exponential form for other real constants.

In exponential form, the real constant is represented in two parts. The part appearing before 'e' is called mantissa, whereas the part following 'e' is called exponent. Thus 0.000342 can be written in exponential form as 3.42e-4 (which in normal arithmetic means 3.42 x 10^{-4}).

Following rules must be observed while constructing real constants expressed in exponential form:

(a) The mantissa part and the exponential part should be separated by a letter e or E.

(b) The mantissa part may have a positive or negative sign. Default sign is positive.

(c) The exponent must have at least one digit, which may be a positive or negative integer. Default sign is positive.

(d) Range of real constants expressed in exponential form is -3.4e38 to +3.4e38.

Ex.: +3.2e-5 4.1e8 -0.2E+3 -3.2e-5

Rules for Constructing Character Constants

(a) A character constant is a *single* alphabet, digit or special symbol enclosed within single inverted commas.

(b) Both the single inverted commas should point to the left. For example, 'A' is a valid character constant, whereas 'A' is not.

Ex.: 'A' 'I' '5' '='

Types of C Variables

A particular type of variable can store only the same type of constant. For example, an integer variable can store only an integer constant, a real variable can store only a real constant and a character variable can store only a character constant. Hence there are as many types of variables in C, as the types of constants in it.

In any C program many calculations are done. The results of these calculations are stored in some cells (locations) of computer's memory. To make the retrieval and usage of these values easy, the memory cells are given names. Since the value stored in each location may change, the names given to these locations are called variable names.

The rules for constructing different types of constants are different. However, for constructing variable names of all types, the same set of rules applies. These rules are given below.

Rules for Constructing Variable Names

(a) A variable name is any combination of 1 to 31 alphabets, digits or underscores. Some compilers allow variable names whose length could be up to 247 characters. However, you should not create unnecessarily long variable names as it adds to your typing effort.

(b) The first character in the variable name must be an alphabet or underscore (_).

Ex.: si_int pop_e_89 avg basic_salary

We should always create meaningful variable names. For example, while calculating simple interest, we should construct variable names like **prin**, **roi**, **noy** to represent Principle, Rate of interest and Number of years, rather than arbitrary variables like **a**, **b**, **c**.

Rules for creating variable names remain same for all the types of primary and secondary variables. So, to help differentiate between variables, it is compulsory to declare the type of any variable that we wish to use in a program. This type declaration is done as shown below.

```
Ex.:  int   si, m_hra ;
      float   bassal ;
      char   code ;
```

C Keywords

Keywords are the words whose meaning has already been explained to the C compiler (or in a broad sense to the computer). There are only 32 keywords available in C. Figure 1.3 gives a list of these 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

Figure 1.3 C keywords.

The keywords should not be used as variable names. However, some C compilers allow you to construct variable names that exactly resemble the keywords.

Compiler vendors provide additional keywords apart from the ones given in Figure 1.3. Though it has been suggested by the ANSI committee that every such compiler-specific keyword should be preceded by two underscores (as in **__asm**), not every vendor follows this rule.

The First C Program

Once armed with the knowledge of variables, constants and keywords, the next logical step is to combine them to form instructions. However, instead of this, we would write our first C program now. Once we have done that, we would see in detail the instructions that it made use of.

The first program is very simple. It calculates simple interest for a set of values representing principal, number of years and rate of interest.

```
/* Calculation of simple interest */
/* Author: gekay  Date: 25/09/2022 */
# include <stdio.h>
int main( )
{
    int   p, n ;
    float   r, si ;
    p = 1000 ;
    n = 3 ;
    r = 8.5 ;
    /* formula for simple interest */
    si = p * n * r / 100 ;
    printf ( "%f\n" , si ) ;
    return 0 ;
}
```

Let us now understand this program in detail.

Form of a C Program

Form of a C program indicates how it has to be written. Though C is free-form language, there are certain rules about the form of a C program that are applicable to all C programs. These are as under:

(a) Each instruction in a C program is written as a separate statement.

(b) The statements in a program must appear in the same order in which we wish them to be executed.

(c) Blank spaces may be inserted between two words to improve the readability of the statement.

(d) All statements should be in lower case letters.

(e) Every C statement must end with a semicolon (**;**). Thus **;** acts as a statement terminator.

(f) Usually, each line contains one statement. However, you can write multiple statements in one line, provided each statement is terminated with a ;.

Comments in a C Program

Comments are used in a C program to clarify either the purpose of the program or the purpose of some statement in the program. It is a good

practice to begin a program with a comment indicating the purpose of the program, its author and the date on which the program is written.

Here are a few tips for writing comments in a C program:

(a) Comments can be in small case, capital or a combination. They should be enclosed within /* */. Thus, the first two statements in our program are comments.

(b) Sometimes it is not very obvious as to what a particular statement in a program accomplishes. At such times, a comment can be used to mention the purpose of the statement(s). For example,

```
/* formula for simple interest */
si = p * n * r / 100 ;
```

(c) Any number of comments can be written at any place in the program. So, a comment can be written before the statement, after the statement or within the statement as shown below.

```
/* formula */  si = p * n * r / 100 ;
si = p * n * r / 100 ;  /* formula */
si = p * n * r / /* formula */  100 ;
```

(d) Comments cannot be nested. This means one comment cannot be written inside another comment. So, following comment is invalid.

```
/* Cal of SI /* Author: gekay date: 25/03/2021 */ */
```

(e) A comment can be split over more than one line, as in,

```
/* This comment has
   three lines
   in it */
```

(f) ANSI C permits comments to be written in another way as follows:

```
// Calculation of simple interest
// Formula
```

What is *main( )*?

main( ) forms a crucial part of any C program. Let us understand its purpose as well as its intricacies.

(a) **main( )** is a function. It is a container for a set of statements. A C program may have multiple functions. If it contains only one function its name has to be **main( )**. All statements that belong to **main( )** are enclosed within a pair of braces { }.

(b) Like functions in a calculator, functions in C also return a value. **main( )** function always returns an integer value, hence there is an **int** before **main( )**. It is known as return type of the function. The integer value that we are returning is 0. 0 indicates success. If statements in **main( )** fail to do their intended work, we can return a non-zero number from **main( )**. This would indicate failure.

(c) The way to watch the value returned by **main( )** varies from one compiler to another, as shown below.

```
Turbo C, Turbo C++ - Alt C | Information
Visual Studio - $ReturnValue in Watch Window of Debugger
Linux - echo $? at command prompt after execution of the program
```

(d) Some compilers like Turbo C/C++ even permit us to return nothing from **main( )**. In such a case we should precede it with the keyword **void**. But this is the non-standard way of writing the **main( )** function. We would discuss functions and their working in detail in Chapter 8.

Variables and their Usage

Let us understand the significance of constants and variables with reference to our program.

(a) Any variable used in the program must be declared before it is used. For example,

```
int   p, n ;              /* declaration */
float   r, si ;           /* declaration */
si = p * n * r / 100 ;    /* usage */
```

(b) In the statement,

```
si = p * n * r / 100 ;
```

***** and **/** are the arithmetic operators. The arithmetic operators available in C are **+**, **-**, *****, **/** and **%**.

printf( ) and its Purpose

C does not contain any keyword to display output on the screen. All output to screen is achieved using readymade library functions like **printf( )**. Let us understand this function with respect to our program.

(a) Once the value of **si** is calculated it needs to be displayed on the screen. We have used **printf( )** to do so.

(b) To be able to use the **printf( )** function, it is necessary to use **#include <stdio.h>** at the beginning of the program. **#include** is a preprocessor directive. Its purpose will be clarified in Chapter 12.

(c) The general form of **printf( )** function is,

```
printf ( "<format string>", <list of variables> ) ;
```

<format string> can contain,

```
%f  for printing real values
%d  for printing integer values
%c  for printing character values
```

In addition to format specifiers like **%f**, **%d** and **%c**, the format string may also contain any other characters. These characters are printed as they are when **printf( )** is executed.

(d) Given below are some more examples of usage of **printf( )** function:

```
printf ( "%f", si ) ;
printf ( "%d %d %f %f", p, n, r, si ) ;
printf ( "Simple interest = Rs. %f", si ) ;
printf ( "Principal = %d\nRate = %f", p, r ) ;
```

The output of the last statement would look like this...

```
Principal = 1000
Rate = 8.500000
```

The output is split over two lines because of newline character '\n'. It sends the cursor to next line. It is one of the several Escape Sequences available in C. These are discussed in detail in Chapter 18.

(e) **printf( )** can print values of variables as well as result of an expressions like 3, 3 + 2, c and a + b * c - d as shown below.

```
printf ( "%d %d %d %d", 3, 3 + 2, c, a + b * c – d ) ;
```

Note that **3** and **c** also represent valid expressions.

Compilation and Execution

Once you have written the program, you need to type it and instruct the machine to execute it. Two other programs are needed to do this—Editor and Compiler. Editor lets us type our program, whereas Compiler converts our program into machine language program. This conversion is necessary, since machine understands only machine language.

Apart from these two, there are other programs which you may need to improve your programming efficiency—Preprocessor, Linker and Debugger. Working with each one of them individually is a tedious job. Hence, often all these are bundled together with a layer of GUI on top of them. GUI makes using these programs easier for you. This bundle is often called Integrated Development Environment (IDE).

There are many IDEs available. Each IDE is targeted towards a particular operating system + microprocessor combination. This combination is known as a platform. A compiler created for one platform does not work with other platforms. Details of which IDE to use, from where to download it, how to install and use it are given in Appendix A. Instead of installing an IDE, there are online options available for compiling and executing programs. These are also discussed in Appendix A.

Receiving Input

In our first C program we assumed the values of **p**, **n** and **r** to be 1000, 3 and 8.5. Every time we run the program; we would get the same value for simple interest. If we want to calculate simple interest for some other set of values then we are required to incorporate these values in the program, and again compile and execute it. This means that our program is not general enough to calculate simple interest for any set of values without being required to make changes in the program. This is not a good practice.

To make the program general, the program itself should ask the user to supply the values of **p**, **n** and **r** through the keyboard during execution. This can be achieved using a function called **scanf( )**. It helps us receive input values them from the keyboard. This is illustrated in the program given below.

```
/* Calculation of simple interest */
/* Author gekay Date 25/09/2022 */
# include <stdio.h>
int main( )
{
    int   p, n ;
    float   r, si ;
    printf ( "Enter values of p, n, r" ) ;
    scanf ( "%d %d %f", &p, &n, &r ) ;
    si = p * n * r / 100 ;
    printf ( "%f\n" , si ) ;
    return 0 ;
}
```

The first **printf( )** outputs the message 'Enter values of p, n, r' on the screen. Here we have not used any expression in **printf( )** which means that using expressions in **printf( )** is optional.

Note the use of ampersand (**&**) before the variables in the **scanf( )** function is necessary. **&** is the 'Address of' operator. It gives the location number (address) used by the variable in memory. When we say **&a**, we are telling **scanf( )** at which memory location should it store the value supplied by the user from the keyboard. The detailed working of the **&** operator would be taken up in Chapter 9.

Note that a blank, a tab or a new line must separate the values supplied to **scanf( )**. A blank is created using a spacebar, tab using the Tab key and new line using the Enter key. This is shown below.

Ex.: Three values separated by blank:

```
1000 5 15.5
```

Ex.: Three values separated by tab:

```
1000	5	15.5
```

Ex.: Three values separated by newline:

```
1000
5
15.5
```

Problem 1.1

Ramesh's basic salary is input through the keyboard. His dearness allowance is 40% of basic salary, and house rent allowance is 20% of basic salary. Write a program to calculate his gross salary.

Program

```
/* Calculate Ramesh's gross salary */
# include <stdio.h>
int main( )
{
    float  bp, da, hra, grpay ;
    printf ( "\nEnter Basic Salary of Ramesh: " ) ;
    scanf ( "%f", &bp ) ;
    da = 0.4 * bp ;
    hra = 0.2 * bp ;
    grpay = bp + da + hra ;
    printf ( "Basic Salary of Ramesh = %f\n", bp ) ;
    printf ( "Dearness Allowance = %f\n", da ) ;
    printf ( "House Rent Allowance = %f\n", hra ) ;
    printf ( "Gross Pay of Ramesh is %f\n", grpay ) ;
    return 0 ;
}
```

Output

```
Enter Basic Salary of Ramesh: 1200
Basic Salary of Ramesh = 1200.000000
Dearness Allowance = 480.000000
House Rent Allowance = 240.000000
Gross Pay of Ramesh is 1920.000000
```

Problem 1.2

The distance between two cities (in kilometers) is input through the keyboard. Write a program to convert and print this distance in meters, feet, inches and centimeters.

Program

```
/* Conversion of distance */
# include <stdio.h>
int main( )
{
    float  km, m , cm, ft, inch ;
    printf ( "\nEnter the distance in Kilometers: " ) ;
    scanf ( "%f", &km ) ;
    m = km * 1000 ;
    cm = m * 100 ;
    inch = cm / 2.54 ;
    ft = inch / 12 ;
    printf ( "Distance in meters = %f\n", m ) ;
    printf ( "Distance in centimeter = %f\n", cm ) ;
    printf ( "Distance in feet = %f\n", ft ) ;
    printf ( "Distance in inches = %f\n", inch ) ;
    return 0 ;
}
```

Output

```
Enter the distance in Kilometers: 3
Distance in meters = 3000.000000
Distance in centimeter = 300000.000
Distance in feet = 9842.519531
Distance in inches = 118110.234375
```

Problem 1.3

If the marks obtained by a student in five different subjects are input through the keyboard, write a program to find out the aggregate marks and percentage marks obtained by the student. Assume that the maximum marks that can be obtained by a student in each subject is 100.

Program

```
/* Calculation of aggregate & percentage marks */
# include <stdio.h>
int main( )
```

```
{
    int  m1, m2, m3, m4, m5, aggr ;
    float  per ;
    printf ( "\nEnter marks in 5 subjects: " ) ;
    scanf ( "%d %d %d %d %d", &m1, &m2, &m3, &m4, &m5 ) ;
    aggr = m1 + m2 + m3 + m4 + m5 ;
    per = aggr / 5 ;
    printf ( "Aggregate Marks = %d\n", aggr ) ;
    printf ( "Percentage Marks = %f\n", per ) ;
    return 0 ;
}
```

Output

```
Enter marks in 5 subjects: 85 75 60 72 56
Aggregate Marks = 348
Percentage Marks = 69.000000
```

Exercises

[A] Which of the following are invalid C constants and why?

'3.15'	35,550	3.25e2
2e-3	'eLearning'	"show"
'Quest'	2^3	4 6 5 2

[B] Which of the following are invalid variable names and why?

B'day	int	$hello
#HASH	dot.	number
totalArea	_main( )	temp_in_Deg
total%	1st	stack-queue
variable_name	%name%	salary

[C] State whether the following statements are True or False:

(a) C language was developed by Dennis Ritchie. T

(b) Operating systems like Windows, UNIX, Linux and Android are written in C. T

(c) C language programs can easily interact with hardware of a PC / Laptop. T

(d) A real constant in C can be expressed in both Fractional and Exponential forms. T

(e) A character variable can at a time store only one character. T

(f) The maximum value that an integer constant can have varies from one compiler to another. T

(g) Usually, all C statements are written in small case letters. T

(h) Spaces may be inserted between two words in a C statement. T

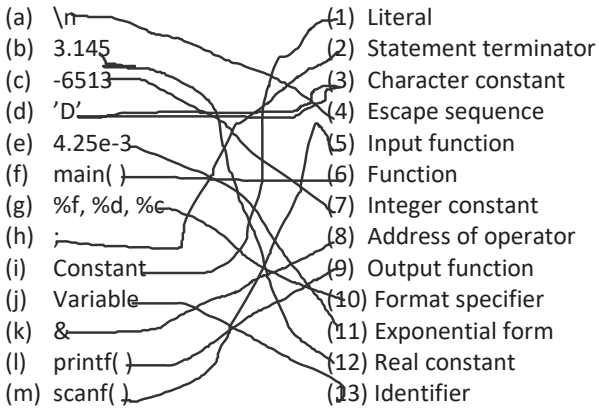
(i) Spaces cannot be present within a variable name. T

(j) C programs are converted into machine language with the help of a program called Editor. F

(k) Most development environments provide an Editor to type a C program and a Compiler to convert it into machine language. T

(l) int, char, float, real, integer, character, char, main, printf and scanf are keywords. F

[D] Match the following pairs:



[E] Point out the errors, if any, in the following programs:

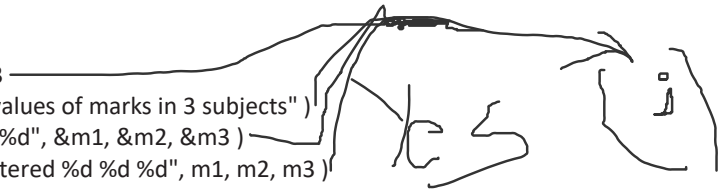
(a)
```
int main( )
{
    int a ; float b ; int c ;
    a = 25 ; b = 3.24 ; c = a + b * b – 35 ;
}
```
No

(b)
```
#include <stdio.h>
int main( )
```

```
{
    int  a = 35 ; float b = 3.24 ;
    printf ( "%d %f %d", a, b + 1.5, 235 ) ;
}
```

(c)
```
#include <stdio.h>
int main( )
{
    int  a, b, c ;
    scanf ( "%d %d %d", a, b, c ) ;
}
```

(d)
```
#include <stdio.h>
int main( )
{
    int  m1, m2, m3
    printf ( "Enter values of marks in 3 subjects" )
    scanf ( "%d %d %d", &m1, &m2, &m3 )
    printf ( "You entered %d %d %d", m1, m2, m3 )
}
```



[F] Attempt the following questions:

(a) Temperature of a city in Fahrenheit degrees is input through the keyboard. Write a program to convert this temperature into Centigrade degrees.

(b) The length and breadth of a rectangle and radius of a circle are input through the keyboard. Write a program to calculate the area and perimeter of the rectangle, and the area and circumference of the circle.

(c) Paper of size A0 has dimensions 1189 mm x 841 mm. Each subsequent size A(n) is defined as A(n-1) cut in half, parallel to its shorter sides. Thus, paper of size A1 would have dimensions 841 mm x 594 mm. Write a program to calculate and print paper sizes A0, A1, A2, ... A8.

KanNotes

- 3 top reasons for learning C :
  - Good base for learning C++, C# or Java later
  - Unix, Linux, Windows, Gaming frameworks are written in C

- Embedded systems programs are written in C

- Constants = Literals -> Cannot change
  Variables = Identifiers -> May change
- Types of variables and constants : 1) Primary 2) Secondary
- 3 types in Primary : 1) Integer 2) Real (float) 3) Character
- Ranges :
  1) 2-byte integers : -32768 to +32767
  2) 4-byte integers : -2147483648 to +2147483647
  3) floats : -3.4×10^{38} to $+3.4 \times 10^{38}$
- In a char constant, both quotes must slant to the left, like 'A'
- Variable has two meanings :
  1) It is an entity whose value can change
  2) It is a name given to a location in memory
- Variable names are case-sensitive and must begin with an alphabet
- Total keywords = 32. Example char, int, float, etc.
- printf() is a function that can print multiple constants and variables
- Format specifiers in printf(), scanf() : int - %i, float - %f, char - %c
- main() is a function that must always return an integer value :
  0 - if it meets success, 1 - if it encounters failure
- void main() is wrong. Correct form is int main()
- Use /* ..... */ or // for a comment in a program
- & is 'address of' operator and must be used before a variable in scanf()