# Computers

## **History**

A successor to the programming language B, C was originally developed at Bell Labs by Ritchie between 1972 and 1973 to construct utilities running on Unix. During the 1980s, C gradually gained popularity.

## **What is Program and why are we are so-called programmers?**

We use application software on devices like mobile devices, smart phones, laptops, tablets, desktops, workstations and servers to name a few everyday.

These softwares consists of one or more programs (Program is a full set of instructions that performs a well-defined task on the device)

Programmers code these **instructions** in a programming language like C, C++.

When a user starts an application program, the operating system loads that program's instructions into part of the RAM and transfers control to the program. The program starts executing, requests data from the user, sends output to the user, and eventually terminates its own execution and returns control to the operating system.

# Information

**Introduction**[**​**](https://intro2c.sdds.ca/A-Introduction/information#introduction)

The information stored in a computer includes program instructions and program data. This information is stored in bits in RAM. The instructions and data take the form of groups of bits. The two most common systems for interpreting information stored in RAM are the binary and hexadecimal numbering systems.

**Fundamental Units**[**​**](https://intro2c.sdds.ca/A-Introduction/information#fundamental-units)

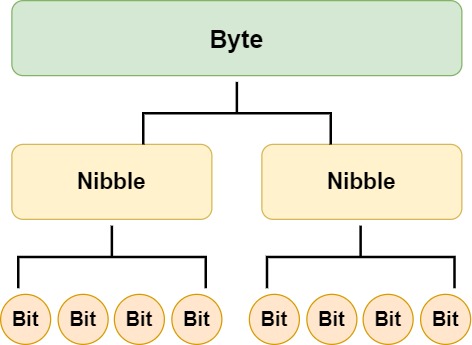
**Bits**[**​**](https://intro2c.sdds.ca/A-Introduction/information#bits)

The most fundamental unit of a modern computer is the binary digit or bit. A bit is either on or off. One (1) represents on, while zero (0) represents off.

Since bits are too numerous to handle individually, modern computers transfer and handle information in larger units. As programmers, we define some of those units.

**Bytes**[**​**](https://intro2c.sdds.ca/A-Introduction/information#bytes)

The fundamental addressable unit of RAM is the byte. One byte consists of 2 nibbles. Each nibble consists of 4 bits.



| **Bit Value (this is 1 Byte)** | **Decimal Value** |
| --- | --- |
|  |  |
|  |  |
| **27  26 25  24  23  22  2**1 **20** |  |
| 0 0 0 0 0 0 0 0 | 0 |
| 0 0 0 0 0 0 0 **1** | 1 |
| 0 0 0 0 0 0 **1**  0 | 2 |
| 0 0 0 0 0 0 **1** **1** | 3 |
| 0 0 0 0 0 **1** 0 0 | 4 |
| ... | ... |
| 0 0 **1** **1** **1** 0 0 0 | 56 |
| ... | ... |
| **1 1 1 1 1 1 1 1** | 255 |

One byte can store any one of 256 (28) possible values in the form of a bit string:

**Hexadecimal**[**​**](https://intro2c.sdds.ca/A-Introduction/information#hexadecimal)

The decimal system is not the most convenient numbering system for organizing information. The hexadecimal system (base 16) is much more convenient.

Two hexadecimal digits holds the information stored in one byte. Each digit holds 4 bits of information. The digit symbols in the hexadecimal number system are {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}. The characters A through F denote the values that correspond to the decimal values 10 through 15 respectively. We use the **0x** prefix to identify a number as hexadecimal (rather than decimal - base 10).

| **Bit Value** | **Hexadecimal Value** | **Decimal Value** |
| --- | --- | --- |
| 00000000 | 0x00 | 0 |
| 00000001 | 0x01 | 1 |
| 00000010 | 0x02 | 2 |
| 00000011 | 0x03 | 3 |
| 00000100 | 0x04 | 4 |
| ... | ... |  |
| 00111000 | 0x38 | 56 |
| ... | ... |  |
| 11111111 | 0xFF | 255 |

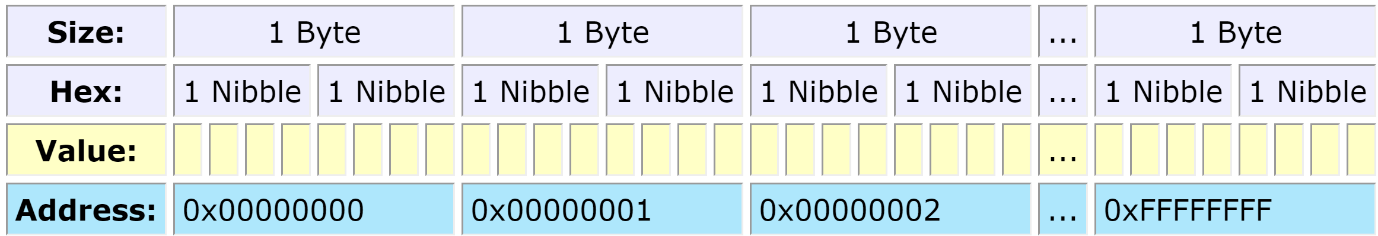
For example, the hexadecimal value 0x5C is equivalent to the 8-bit value 010111002, which is equivalent to the decimal value 92.

**Addresses**[**​**](https://intro2c.sdds.ca/A-Introduction/information#addresses)

Each byte of RAM has a unique address. Addressing starts at zero, is sequential, and ends at the address equal to the size of RAM less 1 unit.

For example, 4 Gigabytes of RAM

* consists of 32 (= 4 \* 8) Gigabits
* starts at a low address of 0x00000000
* ends at a high address of 0xFFFFFFFF



NOTE

Each byte, and not each bit, has its own address. We say that RAM is byte addressable.

**Segmentation Faults**[**​**](https://intro2c.sdds.ca/A-Introduction/information#segmentation-faults) **( you will face it while coding)**

The information stored in RAM consists of information that serves different purposes.

We expect to read and write data, but not to execute it. We expect to execute program instructions but not to write them. So, certain architectures assign the data read and write permissions, while assigning instructions read and execute permissions. Such permission system helps trap errors while a program is executing. An attempt to execute data or to overwrite an instruction reports an error. Clearly, the access has been to the wrong segment. We call such errors a segmentation faults.



## Compilers

**Introduction**

Now the computer only understand thing in bits and bytes.

Anything that human read can be converted to bits and bytes.

For Example:

1 is equal to 00000001.

A is equal to 64 in decimal which is 01000001 in binary.

Now whatever instructions we write and data we provide in programming language needs to get converted to Computer language. The compiler does that for us.



**Programming languages demand completeness and greater precision than human languages.**

**Features of C**[**​**](https://intro2c.sdds.ca/A-Introduction/compilers#features-of-c)

C is one of the more popular third-generation languages. As a procedural language, C requires systematically ordered sets of instructions to perform a computational task and supports the collection of sets of instructions into so-called [functions](https://intro2c.sdds.ca/D-Modularity/functions), which can be accessed from multiple points in the same program as required.

C serves as an excellent, first programming language for several reasons:

* C is English-like
* C is quite compact - has a small number of keywords
* C is the lowest in level of the high-level languages
* C can be faster and more powerful than other high-level languages
* C programs that need to be maintained are large in number
* C is used extensively in high-performance computing
* UNIX, Linux and Windows operating systems are written in C and C++

**Examples**[**​**](https://intro2c.sdds.ca/A-Introduction/compilers#examples)

Let us write a program that displays the phrase "This is C" and name our source file hello.c. Source files written in the C language end with the extension .c.

*/\* My first program // comments introducing the source file*  
 *hello.c \*/*  
  
 #include <stdio.h> *// information about the printf identifier*  
  
 int main(void) *// the starting point of the program*  
 {  
 printf("This is C"); *// send output to the screen*  
  
 return 0; *// return control to the operating system*  
 }

**Linux**[**​**](https://intro2c.sdds.ca/A-Introduction/compilers#linux)

The C compiler that ships with Linux operating systems is called gcc.

To create a binary code version of our source code, enter the following command:

gcc hello.c

**hello.c** -------> gcc --------> a.out

(source code) (compiler) (executable file)

By default, the gcc compiler produces an output file named a.out.

a.out contains all of the machine language instructions needed to execute the program.

To execute these machine language instructions, enter the command:

a.out

The output of the executed binary will display:

This is C

**Documentation**[**​**](https://intro2c.sdds.ca/A-Introduction/compilers#documentation)

We put [comments](https://intro2c.sdds.ca/B-Computations/style-guidelines#comments) to self-document our source code and enhance its readability. Comments are important in the writing of any program.

C supports two styles: multi-line and inline.

C compilers ignore all comments.

**Multi-Line Comments**[**​**](https://intro2c.sdds.ca/A-Introduction/compilers#multi-line-comments)

*/\* My first program*  
 *hello.c \*/*

**Inline Comments**[**​**](https://intro2c.sdds.ca/A-Introduction/compilers#inline-comments)

int main(void) *// the starting point of the program*

**Indentation**[**​**](https://intro2c.sdds.ca/A-Introduction/compilers#indentation)

#include <stdio.h> *// information about the printf identifier*  
  
 int main(void) *// the starting point of the program*  
 {  
 printf("This is C"); *// send output to the screen*  
  
 return 0; *// return control to the operating system*  
 }

By indenting the **printf("This is C")** statement and placing it on a separate line, we show that **printf("This is C")** is part of something larger, which we have called **int main(void)**

**Program Startup**[**​**](https://intro2c.sdds.ca/A-Introduction/compilers#program-startup)

Every C program includes a clause like int main(void). Program execution starts at this line. We call it the program's entry point.

int main(void) *// program startup*  
{  
 return 0; *// return to operating system*  
}

When the users or we load the executable code into RAM (a.out or hello.exe), the operating system transfers control to this entry point. The last statement (return 0;) before the closing brace transfers control back to the operating system.

**Program Output**[**​**](https://intro2c.sdds.ca/A-Introduction/compilers#program-output)

The following statement outputs "This is C" to the standard output device (for example, the screen).

printf("This is C");

The line before **int main(void)** includes information that tells the compiler that **printf** is a valid identifier.

#include <stdio.h> *// information about the printf identifier*

**Case Sensitivity**[**​**](https://intro2c.sdds.ca/A-Introduction/compilers#case-sensitivity)

The C programming language is case sensitive.

If we change the identifier **printf()** to **PRINTF().**The compiler will report a syntax error.