

Unit 3

The C Programming Language: Basic Syntax

Alberto Casagrande

Email: `acasagrande@units.it`

A.Y. 2021/2022

The C Programming Language

The C programming language is an imperative language.

Programs are sequences of instructions.

Trust Your Master

At the beginning, all the programs will have this structure

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    ...

    return 0;
}
```

By the end of the course, you'll have all the details.

Instructions

Are basic “commands” for the computer.

- They are syntactically closed by the symbol “;” (semi-column)
- More than one instruction can lay on the same line

Comments

Even if you are an expert, **unraveling the meaning of a sequence of instructions may be difficult.**

The background features a large, faint watermark of the seal of the University of Trieste. The seal is circular and contains a central illustration of a building with a tower and a sun. The text 'UNIVERSITA' TRIESTE' is visible around the perimeter, and the motto 'VIGILANTIA ET SPLENDOR' is partially visible at the top.

Comments

Even if you are an expert, **unraveling** the meaning of a sequence of instructions may be difficult.

It is really important to comment code

Comments

In ANSI C everything is preceded by `/*` and followed by `*/` is a comment.

It can be longer than one line.

If you come from either C++ or Java, pay attention: single line comment (i.e., `//`) is not standard.

Commento con "/*" e "*/"

```
...  
int y; /* This is a comment  
       and, as you can see,  
       it can include more than  
       one line */  
...
```


Variables

Are “containers” for values.

Each of them is equipped of a **name** and a **type**.

The **name** is used to identify the variable when we want to either “read” or change its content.

The **type** specifies:

- the set of values that can be store into the variable
- the functions that can be applied on them

Declaring a Variable

Any variable must be **declared** before its use.

```
...  
<variable type> <variable name>;  
...
```

From that point on, <variable name> will denote the variable having the specified type.

Variable Names

Variables can have as names any word in

`[a-zA-Z_][a-zA-Z_0-9]*`

excluding **reserved words** (e.g., “int”, “return”)

C is **case sensitive**, e.g., “ciao” \neq “Ciao”

Basic Data Type

Type	Values (at least)	Domain
char	$[-2^7, 2^7 - 1]$ and ASCII Characters	Integer values and Characters
short	$[-2^7, 2^7 - 1]$	Integer values
int	$[-2^{15}, 2^{15} - 1]$	Integer values
long int	$[-2^{31}, 2^{31} - 1]$	Integer values
long long int	$[-2^{63}, 2^{63} - 1]$	Integer values
float	represented by 32 bits	Real values
double	represented by 64 bits	Real values
long double	represented by 128bits	Real values

Basic Data Type

`unsigned` can be used to signal the interest on non-negative values only.

E.g, `unsigned int` variables can assume any value in $[0, 2^{16} - 1]$.

Declaring a Variable

```
/* Integer types */
short s,S; /* at least 16 bits ( $-2^{15}, 2^{15}-1$ ) */
int i,l; /* at least 16 bits ( $-2^{15}, 2^{15}-1$ ) */
long int li; /* at least 32 bits ( $-2^{31}, 2^{31}-1$ ) */
long long int lli; /* at least 64 bits ( $-2^{63}, 2^{63}-1$ ) */

/* Floating point types */
float f; /* 32 bits */
double d,D; /* 64 bits */
long double ld; /* 128 bits */

/* Character and integer type */
char c; /* ASCII or integer values in ( $-2^7, 2^7-1$ ) */
```

Assignments

To store a value in a variable:

```
...  
<variable name> = <expr of the same type>;  
...
```

Assignment Examples

```
int i; char c; short s; float f;  
  
s=4; i=s;  
  
c='a'; /* characters are specified between  
        apostrophes ', while strings  
        between quotation marks " */  
  
i=2*(i+1); /* algebraic expressions are  
           supported (no power) */  
  
f=4.1;  
i=4.1;
```

The last instruction may have weird effects. Why?

Implicit Type Casting

When we assign a value to a variable of different type there is an **Implicit type casting**.

If the the variable's type is more “general”, the value is preserved.

char → short → int → long int → long long int →
float → double → long double

Implicit Type Casting

When we assign a value to a variable of different type there is an **Implicit type casting**.

If the the variable's type is more “general”, the value is preserved.

char → short → int → long int → long long int →
float → double → long double

Otherwise, the value may be approximated.

Explicit Type Casting

Any value can be approximated and forced to a given type by **type casting** it.

```
(<new type>) <expression>
```

E.g.,

```
i=(int)4.1;
```

```
c=(char)((int)c+1);
```

```
i= ((int)(4.1/2))/2;
```

Two Kinds of Division

C implements two different division functions:

- integer division
- floating point division

Their selection depend on the types of operators

```
int i=3, l; float d=3, D;  
  
l=i/2; /* both i and 2 are integers  
      => integer division */  
D=d/2; /* d is a fp variable  
      => floating point division */
```

Boolean Expressions

The supported algebraic relations are:

- equality (`==`)
- majority relations (`>` and `<`) and their derivatives (`>=` and `<=`)
- diversity relation (`!=`)

The supported Boolean operators are:

- logic conjunction (`&&`)
- non-exclusive disjunction (`||`)
- logic negation (`!`)

Boolean Expressions (Cont'd)

Their evaluations are natural numbers:

- 0 represents the Boolean value False
- any other value is interpreted as True

```
char b;
```

```
b = (!(i >= 3) && (d - 2 == -1) && b) || (s != 3);
```

From 1999, C has a Boolean type which is not really used.

Blocks of Instructions

Are either a single instruction or a sequence of instruction between braces “{” “}”

Any variable exists only inside the block in which it is declared.

```
...
{ /* This is a block */
  int h=1;

  h=h+1; /* Here, h does exist */
}

/* Here, it does not */
...
```

Printing on the Standard Output

Use the “instruction” `printf`

```
...
int i=2; long int j=3;
printf("To print numbers: %d %d %d", i, i, j);

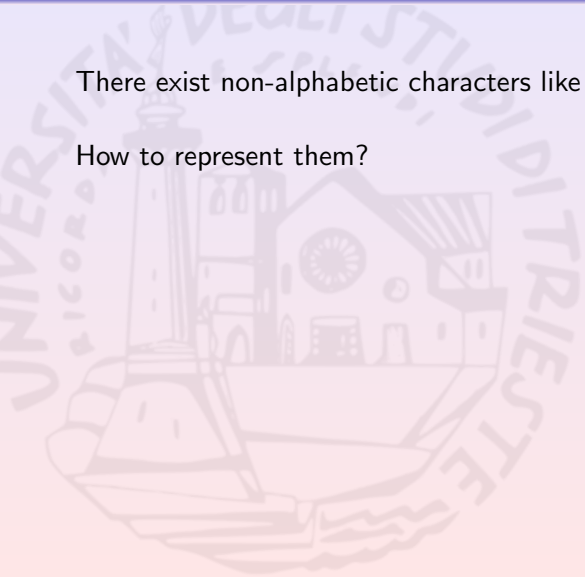
float f=2.2, double d=2.2;
printf("To print floats: %f %f", f, d);

char c='Y';
printf("To print characters: %c", c);
```


Non-Alphabetic Characters

There exist non-alphabetic characters like end line or new line.

How to represent them?



Non-Alphabetic Characters

There exist non-alphabetic characters like end line or new line.

How to represent them? by using **escape sequences**

- `\n` newline
- `\b` backspace
- `\t` horizontal tabulation
- `\\` backslash character
- `\"` double quotation character
- `\a` alert
- `\0` string terminator

Non-Alphabetic Characters (Examples)

```
...  
printf(" This does not end with a \"new line \");  
printf(" This does\nbut this not");  
printf(" This produces an alert sound\a");  
printf(" This is missing the last letter\b");  
printf(" This line e\0nds before its real end");  
...
```

Loop Statements

Blocks of instructions can be repeated many times by using the loop statements:

- while-do
- for

The While-do Statement

Repeats a block while a Boolean condition holds.

```
int i=0;
while (!(i==4)) {

    printf(" Here i=%d\n", i);

    i++; // this is equivalent to i=i+1
}
```

The For Statement

Has the syntax:

```
for (<initialization code>;  
    <loop condition>;  
    <updating code>) <block>
```

E.g.,

```
for (int j=0; j<i; j++) {  
    printf(" Here j=%d\n", j);  
}
```

The Condition Statement If-Then

Executes a block **if and only if** the Boolean condition holds.

```
if (i==2) {  
    printf("i equals 2");  
}
```

The Condition Statement If-Then

Decides the execution of one among two blocks.

```
if (i==2) {  
    printf("i equals 2");  
} else {  
    printf("i differs from 2");  
}
```


Coming next...

- modular programming
- C functions
- libraries

