Master ROC – Bash & C Introduction to C Programming

Sami Taktak

sami.taktak@cnam.fr

September 2023

During the laboratory sessions we will use the GNU/Linux operating system. Start by login under OpenSuse GNU/Linux.

Part 1: Introduction

a) Bonjour le Monde

Lets start with the common example "Hello $_{\square}$ World $_{\square}$!" which display the string "Bonjour $_{\square}$ le $_{\square}$ monde" on its standard output.

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

/* Print the string "Bonjour le monde" on the standard
    output followed by a new line */
printf("Bonjour_le_monde\n");

return 0; // return value of the program
}
```

Listing 1 – C Program « Bonjour le Monde »

- 1. Open a terminal and check that your are actually working in your home folder
- 2. Create a dedicated directory named LabC for this lab session and make it your working directory
- 3. Open a text editor and write the program from listing 1. Save it in a file named hello.c inside the previously created directory. File name of a C file must end with the extension ".c"

To be able to execute that program, we need to compile it. To compile a C program we need to use a compiler. The generic name of a C compiler under UNIX operating systems is cc. Under GNU/Linux, the C compiler is usually provided by the GCC, the *GNU Compiler Collection*.

- 4. Run the command cc hello.c; The program should compile without error
- 5. List the content of the directory LabC; A file named a.out should have been created
- 6. Run the program with the command ./a.out

b) Structure of the program hello

Lets look in more detail to the structure of the program in bonjour.c.

This program bonjour.c has two main parts:

- A preamble that contains here a unique line #include
- The definition of the fonction main

Remarque (Declaration vs. Definition).

- The definition of a function contains:
 - the function name
 - the name and the type of each of its arguments enclosed within parenthesis
 - the type of the return value
 - the body of the function (local variable declarations; list of instruction to be executed)
- A function declaration contains only the prototype of the function which consists of the name, the type of each parameters of the function enclosed in parenthesis ¹ and the type of the return value of the function. This allow the compiler to check that the call to a function is correctly made. For example, the prototype of the main function of the program hello.c would be:

```
// Declaration of the main function int main();
```

• Function main()

The main function is mandatory within a C program. It is the entry point of the program, the first function to be executed. We can that in this simple example, the main function does not take any argument and return the integer value 0.

• Comments

The main function start with 2 comment lines (lines 5 and 6).

In C, comments can be specified:

- either as a block starting with /* and ending with */; Using this syntax, a commend might be spread over many lines
- or as a single line comment starting with // and ending at the end of the line like on line 9

• The printf() function

After the 2 comment lines, we can a call to the function printf(). The printf() takes at least one parameter with consist of the *format string*, which specify what and how to print it. printf() return the number of characters than have been written. That value is ignored in the program hello.c.

The format string allow to describe what needs to be printed and could consist of a single constant string like in $printf("Bonjour_le_lmonde|n")$.

7. Remove the characters '\' and 'n' from the string "Bonjour_le_monde\n". Recompile the program and run it again. What do you notice?

The sequence of characters " \n " is an *escape sequence*. It allow to specify special characters : here the new line character " \n ". The table 1 present the most common escape sequences :

^{1.} the prototype of a function might also specify argument names

Escape Sequence	Corresponding Character
\a	system bip (alert)
\n	new line; line break
\t	tabulation
\\	the anti-slash $(\)$ character itself
\',	simple quote (') character itself
\"	double quote (") character itself

Table 1 – Usual Escape Sequences

• The instruction return

The instruction **return** allow to specify the return value of a function and ends the function execution by returning to the calling function. Here, main() returns the value 0 which, by convention specify that the program has ran without error.

• The directive #include

The directive **#include** allow to include the declaration of the function **printf()**. Without it, the compiler wont be able to know the existence of such a function and would be unable to check if the function call is done correctly.

The printf() function is part of the *Standard C Library* and is always available with the C compiler.

Remarque. We can find the include file needed to call a function from the standard C library by checking the corresponding manual page inside the section 3 of the system manual pages:

```
$ man 3 printf

PRINTF(3) Linux Programmer's Manual PRINTF(3)

NAME
    printf, fprintf, sprintf - formatted output conversion

SYNOPSIS
    #include <stdio.h>
    int printf(const char *format, ...);

:
```

Part 2: Playing with Characters

a) Reading and Writing Characters in C

```
#include<stdio.h>
int main() {
    char c;
```

```
printf("Enter_a_character:_");
c = getchar();

printf("Here_is_the_character:_%c\n", c);
return 0;
}
```

Listing 2 – Reading and Writing Characters in C

- 1. Write the program given in listing 2 into a file
- 2. Compile and execute that program

The program from listing 2 read a character from its standard input and writes it back on its standard output.

We can see that it calls 2 functions from the C standard library :

- the function printf() seen above
- the function getchar() which reads a character from the standard input and return its value, getchar is declared in the header file unistd.h).

The function getchar is called on line 8 and the read value is stored in the variable c.

The variable c is declared on line 5 char c;. It has the type char. The type char is generally used to store character values. In table 2, we have the common type in C;

Remarque. The variable c is declared in the body of the function main() and can only been accessed from within the main() function body (the block enclosed within '{' and '}'). c is a local variable of the function main()).

name	Meaning
char	encoded on 1 byte, allow to store any characters of a 8 bits encoded set
int	allow to store integer values; it size depend of the processor architecture
	(4 bytes on 32 bits architectures, 8 on a 64 bits)
float	single precision floats
double	double precision floats

Table 2 – Basic type in C

On line 10, the function printf() print the string "Here_is_the_character_:." followed by the character and a new line character.

We can see that the position of the read character is specified by a conversion specifier: "%c". "%c" will be replaced by the character encoded by the value specified in c. A format string can contain many conversion specifier, and should be followed by as many argument as there are conversion specifier in the format string.

In the following example, there are 4 conversion specifier and 4 arguments after the format string:

- 3. Write that instruction into a C program; compile that program and run it
- 4. Replace the conversion string "%c" by "%d". Recompile and execute again that program. What do you observe?

The conversion string "%d" specified that the corresponding parameter must be interpreted an integer value. The most common conversion strings are presented in table 3.

String	Meaning
%d	print a integer in base 10
%4d	print a integer in base 10 on 4 digits
%f	print a float
%4.2f	print a float on 4 digits with 2 decimal
%с	print a character
%s	print a string

Table 3 – Conversion strings

b) Character Encoding

A character encoding associate a set of printable characters (digits, letters, graphical symbols, ...) with a set of numerical values. One of the most used encoding is the ASCII (*American Standard Code for Information Interchange*) encoding presented on figure 1.

5. Modify the previous program to display a character together with its encoded value. The function is digit() allow to test if a character is a digit.

```
int is_digit(char c) {
   if ( c >= '0' && c <= '9') {
     return 1;
   } else {
     return 0;
   }
}</pre>
```

Listing 3 – Function returning if a character is a digit

- 6. Modify the previous program so it could tells if a character is a digit or not by using the function is_digit(). Recompile the program and run it
- 7. On line 2 of the function is_digit, digits 0 and 9 are given enclosed within single quotes, why?
- 8. Remove the single quotes, recompile and run the program again. Explain what happen. Fix the program
- 9. Write a function that check if a character is a letter (uppercase or lowercase). Modify the program to check it a character is a digit or a letter

c) Character Array

Array can be declared in C by specifying within square brackets the number of items a variable should contain.

```
type name[size]
```

Each cell of a array can be accessed by specifying the index of the cell we want to have access to. index values start at 0 up to the size of the array -1.

For example, to create a array of 10 integers:

```
int tab_entier[10]; // declaration of an array of 10 integers
int a, b; // declaration of 2 integer a and b
```

```
a = 5;  // affectation of the value 5 to a
tab_entier[1] = a;  /* affectation of the value of a to the cell
of index 1 of the array tab_entier */
b = tab_entier[1];  /* affectation of the value of the cell 1
of tab_entier to b */
```

The following program read a sequence of characters, store them in an array, and finally print the content of the array on its standard output :

```
#include <stdio.h>
 #include <stdlib.h>
 #define SIZE 5
  int main() {
    char tab [SIZE];
    int i = 0;
                       /* index allowing us to count the number
                           of read character initalized to 0 */
    printf("Give_%d_characters_followed_by_Enter:_", SIZE);
    while ( i < SIZE ) { /* loop to read the character (number of
12
                               character to read are given by SIZE) */
13
      tab[i] = getchar();
                                  // incremente the value of i by 1
      i++;
    }
16
17
    // print the table
18
    i = 0;
19
    while (i < SIZE) {
20
      printf("%c", tab[i]);
21
22
    printf("\n");
                               // print a new line
24
25
    return 0;
26
27
```

Listing 4 – Read and write a sequence of characters

- 10. Write, compile et execute the above program. What kind of data structure did we emulate?
- 11. Replace the lines 19 to 24 with printf("%s\n", tab); to use the function printf() to display the content of the array tab as if it was a character string. What happens? Why?

```
printf("Give_\%d_characters_followed_by_Enter:_", SIZE);
    while ( i < SIZE ) { /* loop to read the character (number of
                              character to read are given by SIZE)
12
      tab[i] = getchar();
                                 // incremente the value of i by 1
      i++;
16
    // tab[i]=0;
    // print the table
18
    printf("%s\n", tab);
20
    return 0;
21
22
```

Listing 5 – Read and write a sequence of characters

- 12. Increase the size of the array by 1 (char tab[SIZE+1];) and add the following instruction on line 17: tab[i]= 0;
- 13. Recompile and test the program. It should know behave as expected. Explain why.

Remarque. Pour manipuler des chaînes de caractères correctement, il faut soit connaître le taille, soit être capable de déterminer quand elles se terminent. En C, les chaînes de caractères sont à zéro terminal, c'est à dire que le dernier caractère doit avoir comme valeur numérique 0 pour savoir que la chaîne de caractères est finie.

d) Convert a String to Uppercase

```
#include <stdio.h>
 char upper_case(char c) {
    char m;
    int diff = 'a' - 'A';
    if (c >= 'a' && c <= 'z') {
     m = c - diff;
    } else {
      m = c;
11
12
    return m;
13
14 }
15
16 int main() {
    char c;
17
18
    printf("Enter_a_character:_");
19
    c = getchar();
20
    printf("\%c\n", upper\_case(c));
22
    return 0;
23
```

Listing 6 - Program upper_case

- 14. Write, compile and execute the program of listing 6
- 15. What does it do? Explain line 5 and lines 7 to 11
- 16. Write a function int is_lowercase(char c) which returns 0 if c is a lowercase letter and 1 if not:
- 17. Modify the program upper_case to use the function is_lowercase;
- 18. Modify the program to allow the conversion of a whole string to uppercase

Part 3: Make

a) Makefile

make is a tool to automatize the compilation of a program. It is particularly useful on huge project where the program is split over many files. Only file which needs to be recompiled are recompiled, and make ensure your program is always up to date.

Compilation rules are specified which a file named makefile or Makefile.

The structure of a rule in a makefile is :

```
target : dependencies list
<TAB>command 1
<TAB>command 2
```

The target correspond to the name of the file generated by the associated commands.

dependencies list is the list of dependencies that must be satisfied before running the commands to generate the target. If one of the dependencies are newer that the existing target, the associated commands are being executed. If not, the target is considered up to date and no command is executed.

Remarque. In a general way:

- dependencies list might correspond to files or to other make rules
- target is not necessarily a file. If it is not a file, then the associated commands are always executed

```
all: hello

hello: hello.c
gcc hello.c — o hello

clean:
rm — f hello
```

Listing 7 – Makefile for the program hello

In listing 7, we can see the makefile allowing to compile the program hello if the source file hello.c is newer than the target file hello.

There are 3 rules:

- the first one is the default rules. If make is called without a rule name, the first rule within the makefile is being executed. The default rule is usually called all
- the second rule describe how to build the program hello from the file hello.c
- the third one might be used to clean any file produced by a makefile rule. It usually called clean.

Exercices

- 1. Open a text editor and write the content of listing 7. Save it to a file name makefile in a director named labC;
- 2. Write the program hello.c
- 3. Execute the command make within the directory labC:

```
$ make
gcc hello.c -o hello
$ ls
hello* hello.c Makefile
```

We can see that make automatically runs the compiler to build hello;

4. Run make again

```
$ make
make: Nothing to be done for 'all'.
```

Since hello is up to date against hello.c, make did not rerun the compiler

5. Execute the command make clean:

```
$ make clean
rm -r hello
$ ls
hello.c Makefile
```

- 6. Run the command ls; What do you notice?
- 7. Add a rule in the makefile to compile a program called prog which should be produced out of the prog.c file
- 8. Should we modify the rule all? Should we modify the rule clean?
- 9. How could we tel make to compile only hello or prog?

b) Macro

A makefile could contain macros. They appears at the beginning of the makefile and have the following syntaxe:

```
macro_name = value
```

We often have the macro CC = gcc which specify which C compiler to use. A macro could be called by \$(macro_name).

The shell environment variable are accessible within a makefile as a macro : the macro HOME correspond to the environment variable HOME.

Two other macros are commonly used: CFLAGS and LDFLAGS.

The macro CFLAGS allow to specify the options to send to the compiler.

The macro LDFLAGS allow to specify the options to send to the linker.

Exercices

- 10. Modify the makefile by replacing the call to gcc by a call to the macro \$CC
- 11. Add the option -Wall to the call to the compiler by using the appropriate macro

c) Special Macros

To simplify the ${\tt makefile}$ and to avoid to rewrite similar rules, ${\tt make}$ provide special macros to write generic rules :

— \$@: target name

```
prog: prog.c

$ (CC) prog.c -o $@
```

Listing 8 - makefile rules with generic target

— \$?: dependencies list

```
prog: prog.c fonc1.c
$(CC) $? -o $@
```

Listing 9 - makefile with generic dependencies

— Generic rule : % and \$* can be used to create generic rules. They allow to write a rules that could be applied on any file name. For example, a generic rule to compile a .c file would be :

```
%.o: %.c
$(CC) -c $*.c
```

Listing 10 - makefile with generic rule

Remarque. You can find more information on how to use GNU make at the following URL: http://www.gnu.org/software/make/manual/

Exercices

Let be the following makefile:

```
PHONY: clean

all: test

test: test.o fonc1.o
    gcc test.o fonc1.o —o test

test.o: test.c
    gcc —c —Wall test.c

fonc1.o: fonc1.c
    gcc —c —Wall fonc1.c

clean:
    rm *.o test
```

 $Listing \ 11-{\tt makefile}$

12. Use generic rules to simplify the makefile

Part 4: File copy

Write a program which allow to copy a file into an other file.

To do that, you will use the functions open(), read(), write().

Your program should be composed of multiple function, each of then located in a dedicate source file :

- a function doing the actual copy
- a header file providing the prototype of that functions
- a function main()

Compile and run your program on a text file.

Write the correponding makefile.

Additional question:

Write a function allowing you to parse command line arguments (source file, detination file). You might have a look at getopt() (man 3 getopt) for that;

Part 5: Annexes

77 0x40 115 M 79 0x46 116 M 79 0x46 116 M 81 0x56 120 P 83 0x56 122 R 84 0x56 123 1 85 0x56 124 7 89 0x56 124 7 89 0x56 131 7 90 0x56 133 1 Crochet convrant 91 0x56 133 1 Crochet convrant 92 0x56 134 1 Ancel stantant 94 0x56 134 1 Ancel stantant 94 0x56 141 a Accent circonflexo 96 0x66 140 a Accent circonflexo 96 0x66 140 a Accent circonflexo 101 0x74 141 a Accent circonflexo 102 0x66 <th>Dec</th> <th>Hex</th> <th>Oct</th> <th>Car</th> <th>Signification</th>	Dec	Hex	Oct	Car	Signification
0x4E 1116 N 1 0x4E 1170 Ox4E 1116 N 1 0x50 Ox50 Ox50 Ox50 Ox50 Ox50 Ox50 Ox50	77	0x4D	115	×	
0x4F 117 0 0 0x4F 117 0 0 0x56 120 0 0x57 120 0x56 120 0 0x57 120 0x56 12	78	0x4E	116	N	
0x56 120 0 P 0x51 121 0 P 0x52 123 0 S 0x54 123 123 0 Ox55 125 0 Ox56 125 0 Ox57 125 0 Ox57 125 0 Ox77 127 0 Ox77 0 Ox77 127 0 Ox77	79	0x4F	117	0	
0x55 123 0 0x55 123 0 0x56 123 0 0x56 126 0 0x56 128 0 0x56 128 0 0x56 139 0 0x56 131 7 0x56 133 7 0x56 133 7 0x56 133 7 0x56 134 7 0x67 144 a 0x67 144 a 0x67 156 n 0x67 156 n 0x67 160 p 0x70 160 p 0x71 161 q 0x72 162 n 0x73 170 x 0x74 164 t 0x70 160 p 0x70 160 p 0x71 161 q 0x71 161 q 0x71 161 q 0x72 166 u 0x73 170 x 0x74 164 t 0x77 167 w 0x77 167 x 0x77 167 v 0x77 166 u 0x77 166 u 0x77 166 u 0x77 167 v 0x77 177 167 v	80	0x20	120	۵,	
0x65 122 R 0x64 124 T 0x66 126 U 0x66 126 U 0x66 127 N 0x60 131 T 0x60 133 T 0x60 134 T 0x60 134 T 0x60 140 A 0x60 140 A 0x70 160 D 0x70 160 D 0x70 170 X 0x70 171 A 0x70 D 0x70 D	81	0x51	121	œ	
0x56 123 0 0x56 125 0 0x56 126 0 0x56 126 0 0x56 128 0 0x56 131 0 0x56 132 0 0x56 133 0 0x56 133 2 0x56 133 7 0x56 134 0 0x56 142 0 0x66 140 4 0x67 147 6 0x67 147 6 0x68 150 0 0x68 150 0 0x68 150 0 0x68 150 0 0x67 147 6 0x68 150 0 0x70 155 0 0x70 160 0 0x71 161 0 0x71 161 0 0x72 162 0 0x73 177 0 0x77 167 0 0x77 160 0 0x70 176 1 0x70 177 0 0x77 160 0 0x70 176 1 0x71 161 0 0x71 170 0 0x71 0 0x	85	0x52	122	æ	
0x56 124 U Ox66 126 U Ox66 126 U Ox66 126 U Ox66 131 Z Ox66 130 Cx X Ox66 134 Z Ox66 134 Z Ox66 I Ox66 I Ox60 Z Ox	83	0x53	123	o I	
0x56 125 0 0x66 126 0 0x67 127 W 0x68 131 Y 0x60 132 2 0x60 133 1 0x60 133 1 0x60 138 1 0x60 140 1 0x61 141 a 0x61 143 c 0x61 143 c 0x61 143 c 0x62 143 c 0x63 143 c 0x64 144 d 0x65 146 f 0x66 146 f 0x67 147 g 0x66 150 p 0x67 151 g 0x69 151 g 0x60 156 g 0x70 160 p 0x71 164 g <t< td=""><td>84</td><td>0x54</td><td>124</td><td>⊢ :</td><td></td></t<>	84	0x54	124	⊢ :	
0x56 126 V 0x56 126 V 0x58 133 X 0x56 133 X 0x56 133 X 0x67 134 X 0x67 140 X 0x67 141 A 0x63 143 A 0x64 144 A 0x65 145 B 0x66 146 f 0x67 144 A 0x66 156 f 0x67 147 g 0x66 156 f 0x67 156 f 0x67 156 f 0x67 156 f 0x70 160 g 0x70 160 g 0x71 164 f 0x72 165 g 0x73 171 g 0x76 177 g <t< td=""><td>82</td><td>0x55</td><td>125</td><td>Þ</td><td></td></t<>	82	0x55	125	Þ	
0x65 127 W Ox66 130 Ox66 130 Ox66 130 Ox66 Ox66 Ox66 Ox66 Ox66 Ox66 Ox66 Ox6	98	0x56	126	>	
0x56 130 X 0x56 1310 X 0x56 1310 X 0x56 132 Z 0x56 133 Z 0x56 134 Z 0x67 141 B 0x67 141 B 0x67 144 B 0x67 144 B 0x68 150 B 0x68 151 B 0x68 151 B 0x67 160 B 0x78 117 Z 0x77 167 B 0x78 1170 X 0x78 1170 Z 0x78 Z 0x	87	0x57	127	Μ	
0x66 131 Y 0x66 133 Y 0x66 133 I 0x66 133 I 0x67 134 Y 0x67 140 A 0x63 141 A 0x63 142 B 0x63 144 C 0x66 144 C 0x66 144 C 0x66 144 C 0x66 144 C 0x67 147 B 0x66 146 F 0x67 147 B 0x66 146 F 0x67 152 D 0x68 151 D 0x69 151 D 0x60 154 D 0x70 160 D 0x70 160 D 0x70 160 D 0x70 174 I <t< td=""><td>88</td><td>0x58</td><td>130</td><td>×</td><td></td></t<>	88	0x58	130	×	
0x66 132 Z 0x66 133 Z 0x60 134 \ 0x66 136 \ 0x66 140 \ 0x67 141 a 0x61 141 a 0x62 142 b 0x63 143 c 0x64 144 d 0x65 146 g 0x66 146 g 0x67 150 h 0x68 150 h 0x69 151 j 0x60 155 n 0x66 156 n 0x70 160 p 0x71 161 q 0x72 166 v 0x73 170 x 0x78 177 x 0x70 177 x 0x77 177 x 0x77 177 x <t< td=""><td>68</td><td>0x59</td><td>131</td><td>×</td><td></td></t<>	68	0x59	131	×	
0x65 133 1 0x65 134 1 0x65 134 1 0x66 140 2 0x67 141 a 0x63 143 0 0x63 143 0 0x64 144 d 0x63 143 0 0x64 114 d 0x66 146 f 0x64 152 0 0x67 146 f 0x68 150 p 0x67 145 f 0x60 150 p 0x70 160 p 0x71 161 q 0x70 160 p 0x71 161 q 0x70 170 x 0x71 161 q 0x70 171 x 0x71 164 d 0x60 155 m 0x60 156 m 0x60 156 m 0x71 161 q 0x71 161 q 0x71 161 q 0x72 162 x 0x73 163 x 0x74 164 f 0x77 167 v 0x77 170 x 0x77 170 x	06	0x5A	132	2 '	
0x56 134 \ 0x56 0x56 134 \ 0x56 0x56 134 \ 0x56 136 \ 0x66 140 \ 0x62 142 \ 0x62 143 \ 0x62 143 \ 0x62 144 \ 0x62 144 \ 0x62 144 \ 0x62 144 \ 0x62 146 \ 0x62 156 \ 0x62 166 \ 0x77 167 \ 0x72 162 \ 0x72 162 \ 0x77 167 \ 0x72 163 \ 0x72 164 \ 0x72 167 \ 0x74 164 \ 0x77 167 \ 0x77 177 \ 0x77 177 \ 0x77 \ 0x77 \ 0x77 177 \ 0x77	91	0x5B	133	- .	Crochet ouvrant
0x65 135 0x65 136 0x66 137 0x60 140 0x61 141 0x62 142 0x63 143 0x64 144 0x66 146 0x67 147 0x68 166 0x69 151 0x60 151 0x60 153 0x60 154 0x60 155 0x60 156 0x70 160 0x70 160 0x70 160 0x70 163 0x70 164 0x70 170 0x70 171 0x70 171 0x70 176 0x70 177 0x70 176 0x70 176 0x71 177 0x72 176 0x76 177 0x77 176 <td>92</td> <td>0x5C</td> <td>134</td> <td>/</td> <td>backslash ou antislash (barre oblique inversée)</td>	92	0x5C	134	/	backslash ou antislash (barre oblique inversée)
0x6E 136 0x6C 140 0x6O 140 0x6O 140 0x6S 142 0x6S 143 0x6S 144 0x6S 146 0x6S 146 0x6S 146 0x6S 115 0x7O 110 0x7O 110 0x7O 117 0x7O 117 0x7O 117 0x7O 117 0x7C 117 0x7	93	0x5D	135	_	Crochet fermant
0x66 137 - 0x66 0x61 141 a 0x60 0x61 141 a 0x62 142 b 0x63 143 c 0x64 144 a 0x74 164 a 0x74 164 a 0x77 167 a 0x77 177 b 0x77	94	0x5E	136	<	Accent circonflexe
0x60 1440 0 0x61 1411 0 0x62 143 0 0x63 1442 0 0x64 1444 d 0x66 146 f 0x66 146 f 0x68 150 1 0x68 151 1 0x70 160 p 0x70 160 p 0x71 161 q 0x72 162 x 0x73 163 x 0x73 117 x 0x74 164 0x77 167 w 0x77 117 x 0x78 1170 x 0x78 1170 x 0x78 1171 x 0x78 1171 x 0x78 1171 x 0x79 1171 x 0x77 1171 x 0x77 1171 x 0x77 1171 x	92	OxSF	137	11	underscore (tiret bas ou souligné)
0x65 144 a a Ox66 142 b Ox65 145 b Ox66 145 b Ox66 146 b Ox66 155 b Ox66 155 b Ox66 155 b Ox66 156 b Ox70 160 b Ox70 170 c ox70	96	0x0	140	•	Accent grave
0x63 142 b 0x64 144 d 0x65 144 d 0x66 146 f 0x67 147 B 0x68 150 h 0x69 151 i 0x69 151 i 0x69 151 i 0x60 155 n 0x70 156 n 0x77 161 q 0x77 161 q 0x77 161 x 0x77 161 x 0x78 170 x 0x78 171 y 0x78 171 y 0x77 171 y	97	0x61	141	ಪ .	
0x64 1443 c 0x65 1445 d 0x66 146 f 0x67 147 g 0x68 150 h 0x68 151 h 0x68 151 h 0x68 153 h 0x68 153 h 0x68 153 h 0x68 153 h 0x68 153 h 0x67 155 h 0x70 160 p 0x70 160 p 0x70 160 p 0x71 161 q 0x72 162 h 0x72 163 h 0x73 164 h 0x73 170 x 0x74 164 h 0x77 167 w 0x77 170 x 0x78 177 x 0x78 177 x 0x78 177 x 0x78 177 x 0x78 177 x 0x78 177 x	86	0x62	142	۵	
0x86 1444 d 0x86 146 f 0x86 147	66	0x63	143	o ·	
0x66 146 6 0 0x66 147 8 0 0x69 151 1 0x67 147 8 0 0x69 151 1 0x68 153 1 0x68 153 1 0x66 156 11 0x67 1161 9 0x70 160 0 0x70 163 8 0x77 163 1 0x77 163 8 0x78 110 x 0x78 110 x 0x78 117 x 0x77 117 x 0x78 117 x	100	0x64	144	ø	
0x56 1446 f 0x68 1150 h 0x68 1150 h 0x60 1151 j 0x60 1153 j 0x60 1153 n 0x60 1155 n 0x70 1160 p 0x71 1161 q 0x72 1162 r 0x73 1164 t 0x73 1164 t 0x74 1164 t 0x77 1167 v 0x78 1170 x 0x78 1170 x	101	0x65	145	0	
0x68 150 B 0x68 151 i 0x68 151 i 0x68 153 k 0x66 154 i 0x60 155 m 0x60 156 n 0x70 160 p 0x70 160 p 0x71 161 q 0x72 162 r 0x72 163 r 0x73 163 r 0x73 163 r 0x74 164 r 0x75 110 x 0x77 117 x 0x77 117 x 0x78 177 x 0x78 177 p 0x77 177 p 0x77 177 p 0x77 177 p 0x77 177 p 0x77 177 p	102	99x0	146	¥	
0x68 150 h 0x68 151 i 0x64 152 j 0x66 153 i 0x66 154 1 0x67 156 n 0x70 160 p 0x71 161 q 0x71 161 q 0x72 162 p 0x73 163 s 0x74 164 t 0x75 166 v 0x77 117 v 0x78 177 2 0x78 177 2 0x78 177 2	103	0x67	147	ы	
0x66 151 1 0x68 153 k 0x68 153 k 0x66 154 1 0x60 156 m 0x70 160 p 0x77 161 q 0x72 162 r 0x73 163 r 0x74 164 t 0x77 17 0 1 0x78 170 x 0x77 17 0 1 0x78 170 x 0x77 17 0 1 0x78 170 x 0x77 170 x 0x77 171 y 0x77 y	104	0x68	150	ч	
0x66 152 j 0x66 153 j 0x66 154 l 0x66 156 l 0x67 157 l 0x70 160 p 0x71 161 q 0x72 163 l 0x73 164 l 0x76 166 u 0x76 166 u 0x77 167 u 0x78 170 x 0x70 171 y 0x72 173 z 0x76 174 j 0x77 177 p 0x77 177 p 0x77 174 j 0x77 174 j 0x77 175 z 0x77 174 j	105	69x0	151	ŗ	
0x66 155 k 0x60 156 m 0x60 156 m 0x60 156 m 0x67 157 0 0x77 161 0 0x73 163 x 0x74 164 t 0x77 165 v 0x77 167 v 0x77 170 y 0x78 170 x 0x78 171 y 0x78 171 y 0x77 171 p 0x77 177 DEL	106	0x6A	152	ъ.	
0x6C 1544 11 0x6E 156 nn 0x6E 156 nn 0x6F 157 0 0x70 160 p 0x71 161 q 0x72 162 r 0x73 163 s 0x74 164 t 0x76 166 v 0x77 117 v 0x78 1170 x 0x78 1171 x 0x78 1171 x 0x78 1172 z 0x77 1174	107	0x6B	153	×	
0x6E 1565 m 0x6E 166 m 0x70 160 p 0x71 161 q 0x72 162 r 0x73 163 s 0x74 164 t 0x75 166 v 0x76 166 v 0x77 117 z 0x77 117 z 0x78 170 x 0x78 170 x 0x78 170 z 0x78 170 z	108	0xec	154	п	
0x6E 1566 n 0x70 160 p 0x70 160 p 0x71 161 q 0x73 163 s 0x74 164 t 0x75 165 u 0x76 166 u 0x76 166 u 0x77 170 y 0x77 171 y 0x78 171 y 0x78 171 y 0x70 172 z 0x70 173 f 0x70 174 f 0x70 175 f 0x70 177 p	109	0x6D	155	В	
0x70 160 0 0x70 160 0 0x71 161 q 0x73 163 8 0x74 169 1 0x75 169 1 0x77 167 v 0x77 117 2 0x78 172 z 0x78 172 z 0x78 173 (0x70 171 2 0x70 171 2 0x71 172 z 0x77 172 z	110	0x6E	156	п	
0x70 1660 p 0x71 1610 p 0x72 162 r 0x73 166 u 0x76 166 u 0x76 166 v 0x77 170 x 0x78 177 w 0x78 171 y 0x70 172 z 0x70 174 0x70 174 0x70 174 0x70 174	111	0x6F	157	0	
0x72 162 q 0x72 162 s 0x72 163 s 0x74 164 u 0x76 166 u 0x77 167 w 0x78 170 x 0x78 171 x 0x78 172 z 0x78 172 z 0x78 172 z 0x78 172 z 0x78 177 p 0x77 174 [112	0x20	160	Q,	
0x72 162 r 0x73 163 r 0x74 164 t 0x75 166 u 0x77 167 w 0x77 177 y 0x78 172 z 0x78 172 z 0x78 173 { 0x77 173 { 0x77 174 0x76 174 0x77 174	113	0x71	161	ם,	
0x73 163 s 0x74 164 t 0x75 166 u 0x76 166 v 0x77 167 v 0x78 170 x 0x78 171 y 0x70 172 z 0x70 173 { 0x70 174 { 0x70 175 }	114	0x72	162	н	
0x74 164 t 0x75 166 u 0x76 166 v 0x77 167 w 0x79 171 y 0x74 172 z 0x70 174 0x7C 174 0x7C 174 0x7C 174 0x7C 177 DEL	115	0x73	163	o ₂	
0x75 165 u 0x77 167 w 0x77 167 w 0x78 170 x 0x78 171 y 0x70 172 z 0x70 174 0x70 174 0x70 175	116	0x74	164	¢	
0x77 167 v 0x77 167 v 0x78 170 x 0x78 171 2 0x78 173 { 0x70 175 2 0x70 176 } 0x70 176 }	117	0x75	165	n	
0x77 167 w 0x70 0x78 170 x 0x79 171 y 0x74 172 z 0x78 173 4 0x70 174 0x70 176 - 0x7E 177 DEL	118	0x76	166	٥	
0x78 170 x 0x79 171 x 0x78 172 z 0x78 173 { 0x7C 174 { 0x7C 176 } 0x7E 177 DEL	119	0x77	167	В	
0x79 171 y 0x7A 172 z 0x7B 173 { 0x7C 174 0x7C 176 } 0x7E 177 DEL	120	0x78	170	×	
0x7A 172 z 0x7B 173 { 0x7C 174 0x7D 175 } 0x7F 177 DEL	121	0x79	171	Α	
0x7B 173 { 0x7C 174 0x7D 175 } 0x7D 176 - 0x7F 177 DEL	122	0x7A	172	Z	
0x7C 174 0x7D 0x7D 175 } 0x7E 176 ~ 0x7E 177 DEL	123	0x7B	173	ب	Accolade ouvrante
0x7D 175 } 0x7E 176 ~ 0x7F 177 DEL	124	0x7C	174	_	Barre verticale
0x7E 176 ~ 0x7F 177 DEL	125	0x7D	175	<u>~</u>	Accolade fermante
0x7F 177 DEL	126	0x7E	176	ŧ	Tilde
	127	0x7F	177	DEL	Delete (effacement)

Signification	Null (nul, inexistant)	Start of Header (début d'en-tête)	Start of Text (debut du texte) End of Text (fin du texte)	End of Transmission (fin de transmission)	Enquiry (End of Line) (demande, fin de ligne)	Acknowneage (accuse de reception) Bell (caractère d'appel)	Backspace (espacement arrière)	Horizontal Tab (tabulation horizontale)	Line Feed (saut de ligne) Vertical Tab (tabulation verticale)	Form Feed (saut de page)	Carriage Return (retour chariot)	Shift Out (fin d'extension)	Shift In (démarrage d'extension)	Davice Control 1 to 4 (DC1 et DC3 sont	généralement utilisés pour coder XON et	XOFF dans un canal de communication	duplex) Nonativo Astromisdae (accusé de récention négatif)	Negative Acknowledge (accuse de reception negatit) Simchronous Idle	End of Transmission Block (fin du bloc de transmission)	Cancel (annulation) End of Medium (fin de sunnort)	Substitute (substitution)	Escape (échappement)	File Separator (séparateur de fichier)	Group Separator (Separateur de groupe) Record Separator (Séparateur d'enregistrement)	Unit Separator (séparateur d'unité)	Space (espace)	Foint d'exclamation Guillemet droit	Dièse ou signe numéro	Dollar	Fourcent Renorlingto ou nordusto	Apostrophe ou guillemet fermant simple ou accent aigu	Parenthèse ouvrante	Parenthèse fermante	Plus	Virgule	Moins on tiret on trait d'union	Foint Stash (barre oblique)									Deux-points	Point-virgule Inférieur	Égal Égal	Supérieur		Arobase ou A commercial									
Car	NOL	SOH	STX	EOT	ENG	BEI.	BS	TAB	i i	FF	CB.	SO	SI	DC1	DC2	DC3	DC4	SYN	ETB	CAN	Sign	ESC	FS	3 2	ns	SP -	:	#	↔ ;	~ &	8 ^	J	^ ÷	+	•	1	. \	. 0	π.	0 0	n 4	ß	91	- 00	6		·- v	, II	^	٠. و	9 4	м	D I	οы	I II	0	≖ ⊢	י י	₩.	1
Oct	000	001	002	004	002	000	010	011	012	014	015	016	017	020	022	023	024	026	027	030	032	033	034	036	037	040	041	043	044	045	047	020	051	053	054	055	057	090	061	062	064	990	990	040	071	072	073	075	920	770	101	102	103	104	106	107	110	112	113	****
Hex	00×0	0x01	0x02 0x03	0x04	0x05	0x00	0x08	60x0	OxOR	0x0C	0x0D	0x0E	OxOF	0x10	0x12	0x13	0x14	0x15	0x17	0x18	0x1A	0x1B	0x1C	0x1D	0x1F	0x20	0x21	0x23	0x24	0x25	0x26	0x28	0x29	0x2B	0x2C	0x2D	OX2E Ox2F	0x30	0x31	0x32	0x33	0x35	0x36	0x3/	0x39	0x3A	0x3B	0x3D	0x3E	0x3F	0x40 0x41	0x42	0x43	0x44 0x45	0x46	0x47	0x48	0x4A	0x4B	25.40
Dec	0	₩ (01 m	4	ю¢	٥ ٢	. 00	о :	10	12	13	14	15	17	18	19	20 5	77	23	24	56	27	5 28	30	31	32	34	38	36	37	9 66	40	41	43	44	45	46	48	49	20	52	53	24	26	22	28	20	61	62	63	65	99	67	89 69 99 99	02	7.1	7.5	74	75	2

FIGURE 1 – Table de caractère ASCII