

Live Museum – Museum of Computer Science
Escola Tècnica Superior d'Enginyeria Informàtica¹
Universitat Politècnica de València²
RETROCOMPUTING WORKSHOP

Jorge González

jgonzalez@dsic.upv.es

Have you ever heard about 8bit computers?

And about Golden Age of Spanish Software?

We are certain that this experience will enrich your perspective as modern computing user, we assure you that you will enjoy it, you may be for a while in the shoes of a BASIC programmer from 80s-90s decade.

Of course, after this workshop you will be able to answer the questions given above.

1. Introduction

The Live Museum's classroom consists of several Amstrad CPC from the 80s, more precisely, some 464 and 6128 models. In our context, the difference between these two types of CPC is absolutely obvious: while 464 have got a cassette player built in, 6128 includes a built-in 3" disk drive instead.

Even though there are both Spanish and English CPC versions, the essential difference can be found in keyboard configuration and the name of some keys. Furthermore, the distinction between a national system and an international one is completely irrelevant.

Finally, comparing again both CPC versions, we could distinguish between computer equipment supplied by its own monitor (colour monitor or green screen), but also computers with an external supply as TV, thus, an external power source is needed. This is an important feature to know how starting up the system and shutting it down.

2. Turning the computer Off/On

Different features:

- CPC Amstrad Equipment with Amstrad monitor (colour or green screen)
 - Turn it ON/OFF directly from the monitor. Usually, the computer switch should be in the ON position. So, turning ON/OFF the monitor will be enough to turn on the whole computer. If it does not work maybe the computer switch ON/OFF is in the OFF position. In this case move it in order to turn ON the computer.

¹ Higher Technical School of Computer Engineering

² Polytechnic University of Valencia

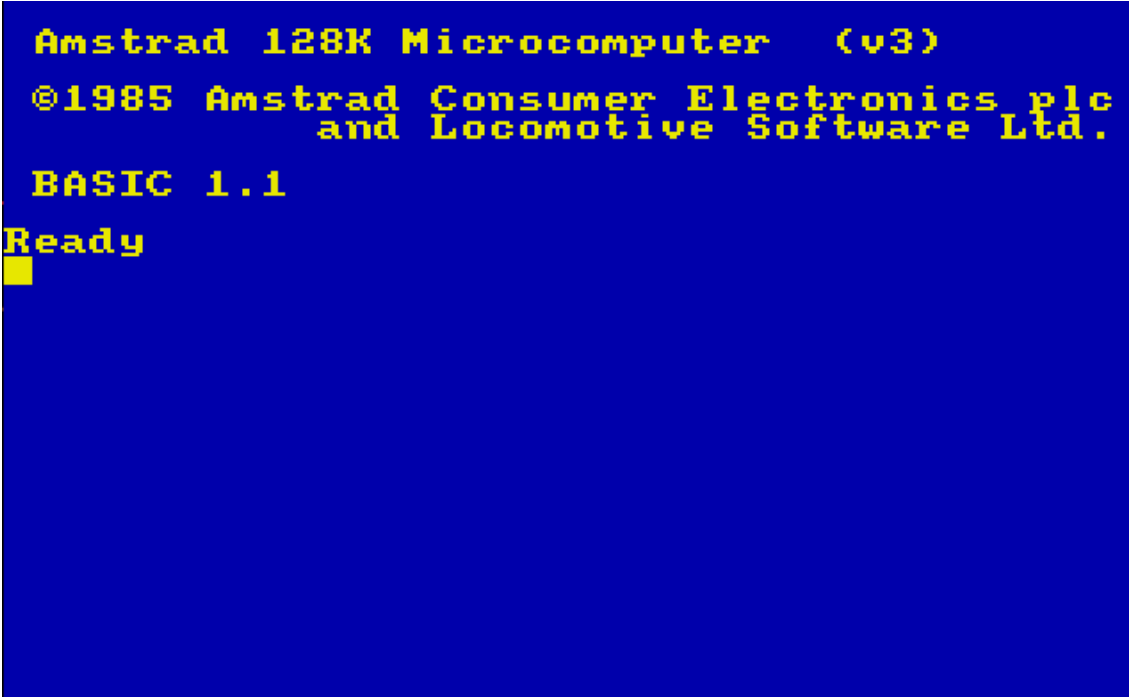
- CPC Amstrad Equipment with TV. According to their external power supply:
 - Amstrad power supply. Video Output is tuned by RF onto TV channel 0.
 1. Turn on the TV. By default, TV goes automatically to channel 0. If it does not, then switch on channel 0.
 2. Turn on the computer by means of its power switch.
 3. Separately, turn off both the computer and the TV.
 - Generic power supply. Audio and video outputs are connected via SCART.
 1. Turn on the TV. External input signal is not automatically detected.
 2. Switch TV program to SCART input.
 3. Turn on the computer by means of its power switch.
 4. Separately, turn off both the computer and the TV.

Finally, in order to reset the computer equipment, the keyboard combination below could work: CONTROL/CTRL+MAYS/SHIFT+Esc. If it does not work, you can use the computer switch.

3. OK, the computer is working so... what should we do next?

First of all, notice how quick these computers start working. It is an instant-on, ready to receive orders, nothing to do with nowadays computers, even mobile phones or tablets take more time to start up!

Those yellow letters on a blue background (if your monitor is not a monochromatic one) are a welcome message loaded from ROM (read-only memory). For datacoders, or CPC464s, text is a bit different, mainly because they are 64K of RAM and 1.0 of BASIC version.

A screenshot of the Amstrad 128K Microcomputer boot screen. The background is a solid blue color, and the text is displayed in a yellow, monospaced font. The text is arranged in several lines: the first line reads 'Amstrad 128K Microcomputer (v3)', the second line reads '©1985 Amstrad Consumer Electronics plc and Locomotive Software Ltd.', the third line reads 'BASIC 1.1', and the fourth line reads 'Ready'. A small yellow square cursor is positioned at the beginning of the 'Ready' line.

```
Amstrad 128K Microcomputer (v3)
©1985 Amstrad Consumer Electronics plc
and Locomotive Software Ltd.
BASIC 1.1
Ready
█
```

Ready message at the end indicates that the system is already available to receive commands from the keyboard.

3.1. Interpreter of BASIC

The interpreter of BASIC is a software stored in ROM, and it is executed immediately after you start the computer, the message above is part of this software. The square shaped cursor appears below the **Ready** message, any text typed next will be in the position of the cursor on the screen, and you can send any order to the system finishing by means of the keys **INTRO**, **ENTER** or **RETURN**.

Try on now to write **PRINT** 'Hello World' on the interpreter of BASIC and press **INTRO**

You just ran the command **PRINT**, which displays on the screen the message 'Hello World' indicated below, its execution is immediate due to the fact that it is an interpreter. As you can see, when you press **INTRO**, text 'Hello World' is displayed on the screen, after this text again the word **Ready** encouraging us to execute the next order.

```
Amstrad 128K Microcomputer (v3)
©1985 Amstrad Consumer Electronics plc
and Locomotive Software Ltd.

BASIC 1.1

Ready
PRINT "Hello World"
Hello World
Ready
■
```

Interpreter immediately responds to the order we give to the system, in this way we can always see the behavior of the order at the moment. Nevertheless, if we want something bigger it is not very useful, in other words: if we want to supply the computer with a major complexity behavior, what we need is creating a program, an execution routine which the computer has to follow step by step, in sequence and unambiguously.

The infrastructure required for software development is standard in BASIC. For that purpose, you have to write the order you want preceded by a number.

Now write **10 PRINT** 'Hello World' in the **BASIC** interpreter and press **INTRO**

Nothing has happened here, right? The cursor just moved with a line break, as in every text editor. In fact, not even **Ready** message appeared, although the system is as ready as it was before. What happened is that this order has been incorporated to the user program, you can access to it writing **list** (and remember pressing **INTRO** at the end) in the BASIC interpreter, and you can run it as many times as you like through the **run** command of BASIC.

Write **LIST** in the Basic interpreter and press **INTRO**; then write command **RUN**

```
Amstrad 128K Microcomputer (v3)
©1985 Amstrad Consumer Electronics plc
and Locomotive Software Ltd.

BASIC 1.1

Ready
PRINT "Hello World"
Hello World
Ready
10 PRINT "Hello World"
list
10 PRINT "Hello World"
Ready
run
Hello World
Ready
run
Hello World
Ready
■
```

A program is a numbered sequence of BASIC instructions which are executed in order. Why the first line of our program have been numbered with 10 instead of 1? Easy, a program is created little by little and it has variable features over time, so once a part is written, maybe you would need adding a new code line, for instance, at the very beginning. If line 1 is already taken, the only way to do that is by renumbering every single line. To avoid this problem, lines are usually numbered in groups of ten, in order to leave room among them for adding new instructions later.

3.2. BASIC Programs

Now write the program below:

```
10 REM Even: CLS
20 INPUT "Give me a number: "; N
30 R = N MOD 2
40 IF R=0 THEN PRINT "It is even"
50 GOTO 20
```

When you have finished, try to execute it by means of the BASIC command **run**. If you want to edit any line, for instance fixing a mistake, just write **edit**, press space bar, and then write the number of the line you want to edit. This process permits you editing the content of the line you indicated before.

e. g. Write **EDIT 10** in the BASIC interpreter (then press **INTRO**) and now edit line 10

Now, we are going to explain you a bit more about the previous program. First of all, you should know that in BASIC only 1 command per program line is usual. This feature makes easier its own reading, maintenance and sustainable development. However, BASIC's syntax permits to incorporate several instructions inside one program line. In that way it behaves as if they were in different lines, in other words, these commands are executed in order, from left to right, and the

syntax of BASIC demands that the punctuation mark `:` has to be the dividing element between instructions. In fact, line 10 from previous program contains two instructions (**REM** and **CLS**). The **REM** command of BASIC makes possible introducing a comment in order to help with the reading of the program, which in that case, it is only being used to add a title. The **REM** command does not imply execution; it is only a comment inside the program. On the other side, **CLS** (clear screen) just has the task of clearing the screen.

Edit the program leaving only 1 instruction per line (leave **CLS** on new line 15)

Line 20 (**INPUT** command) allows asking the user (text surrounded with quotation marks), whose answer will be saved in execution time inside the variable named N^3 . Notice the symbol `;` to separate the user question from the variable that its answer will have.

Then, at line 30, the introduced number (remember, variable N) is divided by 2 with the **MOD** command, the rest of the division will be calculated after its execution. The result will be saved in the variable **R**.

Eventually, in line 40 you indicate the system to take into account the content of variable **R**. If the content of **R** is 0 a message is displayed. On the contrary, if **R** contains a different value (not 0), then you do not have to do anything.

In line 50, at the end of the program (**GOTO**), the order becomes disrupted forcing BASIC to make a gap, going back to the indicated line. In this case, **GOTO 20** command avoids the program ending in a regular way, so every time execution arrives to line 50 it goes back to line 20 asking and answering the user again and again.

To stop the virtually endless execution of the program, press **Esc** at least twice

There are more elegant ways of designing loops (code section repeated). Please, now write next program, which uses the **FOR** instruction of BASIC:

Before, write **NEW** in the BASIC interpreter in order to delete the previous program

```
10 REM Example
20 CLS
30 INPUT "Give me N: "; N
40 FOR I=1 TO N
50 PRINT I, SQR(I)
60 NEXT I
```

As you can imagine, the user puts a value of N to the system's question, and there is a loop then (lines 40-60) defined as a counter from 1 to N value, so in every single iteration the variable **I** will

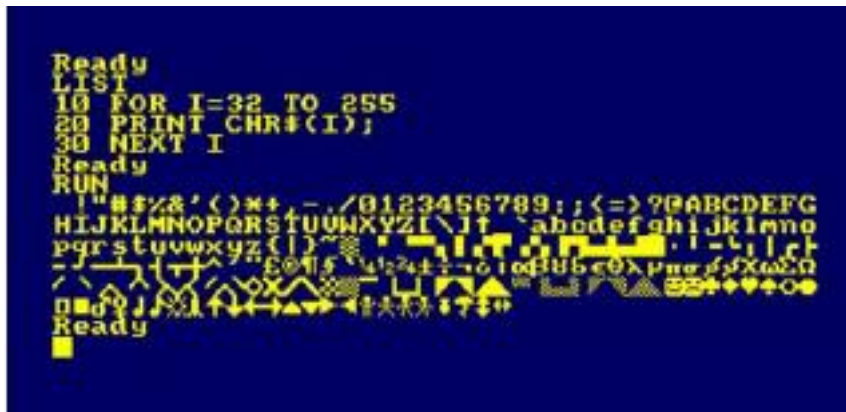
³ A variable is like a box used to store only one datum (Section 3.2.5)

take a different value from the counter, for that reason the screen will display the list of square roots of the integers from 1 to N.

3.2.1. Character set

The character set of BASIC is a set of letters, numbers and symbols you can use by means of the keyboard or defined functions for text management. If you want to see all the printable character set of Amstrad's BASIC, just write the program below (you do first **NEW** to clear previous program):

```
10 FOR I=32 TO 255
20 PRINT CHR$(I);
30 NEXT I
```

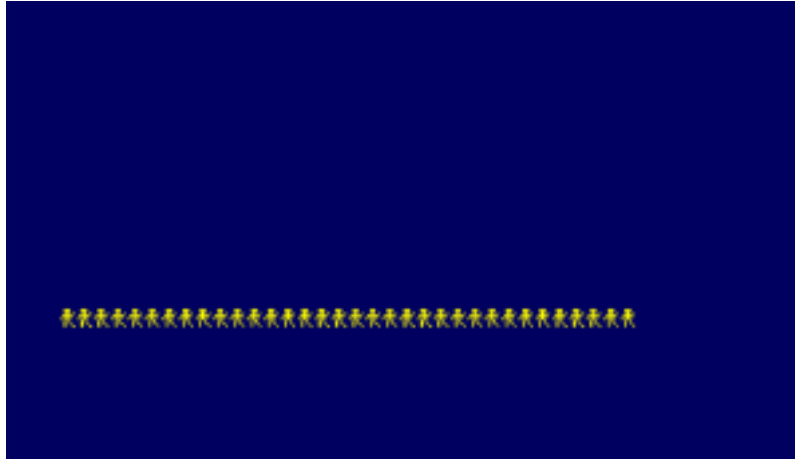


The Amstrad's character set uses only 1 byte (8 bits) to manage $2^8=256$ characters, which are numbered (like almost everything in computer science) starting from 0, that is, from 0 to 255. The program shows almost every character of BASIC but the first 32 which are checking characters. Those ones, due to their nature, are not able to be printed through the monitor. Symbol ; at the end of line 20 is there to avoid the line break after printing each character.

3.2.2. The man who walks

The screen is in 1 mode by default, 40 columns and 25 lines, all of them numbered in increasing order, starting from 1, and from the upper-left corner. You can use the command **LOCATE** to place the cursor on the desired coordinate. Its syntax is **LOCATE, N° COLUMN, N°ROW**, as shown below:

```
10 CLS
20 FOR X=1 TO 40
30 LOCATE X, 20
40 PRINT CHR$(250)
50 NEXT X
60 GOTO 10
```



Edit now line 40 by means of the **EDIT** command and change it for the text below:

```
40 PRINT " "; CHR$(250)
```

What happens next? Could you explain the reason why it gives you a false sense of movement? Notice how line 40 looks like, we keep painting the walker, but before that, we put a blank space in order to overwrite the walker from the previous iteration, thus erasing the walker gives you this sense of movement.

3.2.3. Graphics

In addition, there is a graphic mode in the screen where resolution is 640x400 pixels. Unlike text mode, the beginning of coordinates is located in the lower-left corner. Now, write the following program to draw a rectangle, framing the screen to 10 pixels:

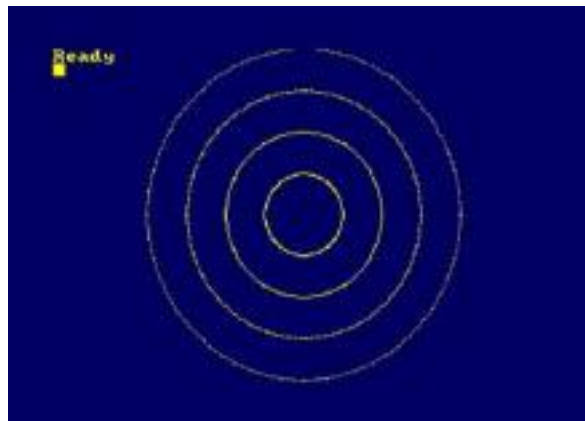
```
10 CLS
20 PLOT 10,10
30 DRAW 10,390
40 DRAW 630,390
50 DRAW 630,10
60 DRAW 10,10
```



It is also possible nesting a loop inside one another. Thus, a loop drawing a circle could be inside one another which will vary its radius within a range in every single iteration.

Now, type the following program to draw concentric circles on the center of the screen:

```
10 CLS
20 DEG
30 FOR R=50 TO 200 STEP 50
40 ORIGIN 320,200
50 FOR A=1 TO 360
60 PLOT R*COS(A),R*SIN(A)
70 NEXT A
80 NEXT R
```



3.2.4. Sound

An integrated speaker is standard on Amstrad CPC systems, even though they also have an audio output by means of a Mini Jack connector, which can serve to bring sound to the TV speakers via Scart.

The BASIC command to produce sound is **SOUND**, describing channel and pitch period. As an optional feature you can add length, volume, surrounding pitch and volume, and period of noise.

There are three sound channels (1, 2 and 4), that is what permits simultaneous sounds, as well as the ability of sending out the same musical note through several sound channels at the same time. For that, we have to calculate the sum of the channels we are going to use, and put the result in the **SOUND** order, so if we want to play the sound through e.g. both channel 1 and 4 we will put 5, or if we want to hear it through all the channels at time, we will put 7 (1+2+4=7). On the other hand, we can specify the musical note we want to hear through its pitch period, which is encoded as follows (notes in an intermediate scale from 8 available in Amstrad CPC):

Pitch Period	Musical Note
478	DO
426	RE
379	MI
358	FA
319	SOL
284	LA
253	SI

So, these are instructions if you would like to play the note DO on channel 1: **SOUND 1, 478**. The length of the sound is 0.2 seconds by default, almost unnoticeable. If you want to vary the length, you could add a third parameter to **SOUND** (measured by hundredths of a second). Next parameter would be the volume level (an integer between 0, silence, and 7, max.). Now, type this command of BASIC to hear the note DO, at a medium level volume during 2 seconds:

```
SOUND 1, 478, 200, 4
```

Try to create a program with a melody you already know, or if you do not, write the one provided below (even if you do not understand all the instructions), and try to guess what it sounds like.

```
10 tempo=2.5
20 RESTORE 90
30 FOR x=1 TO 37
40 READ pitch, length
50 freq=440*(2↑(0+ ((pitch-10)/12)))
60 pitchnum=ROUND (125000/freq)
70 SOUND 1,pitchnum,length*tempo,7
80 NEXT
90 DATA 27,10,29,10,25,10,22,20,24,10,20,20
100 DATA 15,10,17,10,13,10,10,20,12,10,8,20
110 DATA 3,10,5,10,1,10,-2,20,0,10,-2,10,-3,10,-4,40
120 DATA 3,10,4,10,5,10,13,20,5,10,13,20,5,10,13,40
130 DATA 13,10,15,10,17,10,13,10,15,10,17,20,12,10,15,20,13,40
```

3.2.5. Variables

Variables are an essential part of development in all BASIC programs, as well as in any other programming language. Variables are like boxes where we can store only one item, so you can use their content to execute any operation later. For that, you have to identify them with a name or tag which you can identify without any ambiguity. The valid names start with a letter followed by more letters or numbers (or some symbols as well, such as the underlined symbol), without blank spaces, international characters or punctuation marks.

```
10 X = 2
20 PRINT "Variable X contains: "; X
```

Immutability is one of the main features of variables in programming, in other words, if you do not overwrite a different information, the content of the variable keeps the same value indefinitely.

```
30 X = 5
```

```
40 PRINT "Now the variable X contains: "; X
```

If you want the user to enter a value for the variable, use command **INPUT**:

```
10 INPUT X
```

```
20 PRINT "Variable X contains: "; X
```

```
30 PRINT "Its square is: "; X*X
```

Command **INPUT** will display a question mark to inform the user that the system is waiting for a piece of information. For instance, if the user enters 3, on the screen you will see:

```
?3
```

```
Variable X contains: 3
```

```
Its square is: 9
```

Command **INPUT** enables you to include text before the question mark:

```
Edit line 10 of your programme: 10 INPUT "WHAT NUMBER YOU WANT TO SQUARE"; X
```

Another property of variables in any programming language is their data type, that is to say, which type of information they can work with, so storing it. In general, variables behave like boxes (but a particular kind of), so their content depends on the data type they were defined to, in the same way a shoebox only serves to store shoes (theoretically).

As they have been used until now, BASIC variables define boxes containing real numbers, but in programming there is another, very usual, data type, strings. It consists in a sequence of characters we use to write messages. In order to define a text variable, at odds with its definition as numerical type, you only have to use an identifier which must end with character \$:

```
10 INPUT "Enter your name: "; NAME$
```

```
20 SENTENCE$ = "HELLO " + NAME$
```

```
30 PRINT SENTENCE$
```

Notice that bracketed delimiters are used with string literals, avoiding a potential syntax error when running the BASIC program, since otherwise string literals are interpreted as variables. You can also observe how to concatenate two strings by means of symbol + in line 20.

3.2.6. Suggested programs

1. Build a program which requests user data:

- User name
- User year of birth
- Current year

Then the program will display a custom sentence for the user, saying hello by their names, and indicating user's current age (or the age they will reach the current year).

2. Broaden the previous exercise indicating the length of user's name (function **LEN** (string) returns length or number of characters of the string given between parentheses).

3. Write a programme which asks the user to write any 2 strings of characters, and then displays on the screen which one has a larger number of characters (remember conditional instructions of BASIC: **IF** condition **THEN** command).
4. Write a programme which asks the user to write any 3 strings of characters, and then displays on the screen which one has a larger number of characters (first, you compare strings 1 and 2 and determine which one is larger than the other, then you compare the previous winner with string 3, so the winner will be the largest one).
5. Write a programme which asks the user to write any **N** strings of characters, and then displays on the screen which one has a larger number of characters (user is initially required to provide **N**). Think how this exercise can be based on the previous one, in order to generalise the procedure by using a **FOR** loop of repetition. For that, the best option is starting from an empty string "" (with 0 characters) as the first winner string, and then reading a new string in the loop, comparing it with the current winner string to find out which one is larger than the other. The result may be the winner being the same or that it has to be updated. Remember about immutability of variables and conditionals of BASIC.
6. Change the previous program and calculate the average length of introduced strings. First of all, calculate the summation of **N** string lengths, which is usually solved in programming through accumulator variables, meaning, these variables are overwritten in each iteration of the loop depending on its own old value. In summation cases, these variables are initialized with the neutral value of the sum, **V=0**. Then, a new value **D** for the summation will be computed in each iteration, so it will be added to the current content of the accumulator variable **V**, producing a partial summation, then you save it in the same accumulator variable, **V = V+D**.
7. Create a *Guess the number* program. In the line 10 fix a variable to an integer among 1 to 100, a friend of yours will have to guess it with a maximum of 6 attempts. Your friend will write a number on each turn, and the program will say if the secret number is bigger or smaller, if it is the right number the game is over. Remember command **GOTO N°**. You can use the well-known counter variables, which is a particular case of accumulators where **D** is always 1. Anyway, probably your neighbour will have a different program, so... Learn as much as you can from different solutions provided!