Coursework 2: Basic Programming

Héctor Menéndez[1]

AIDA Research Group
Computer Science Department
Universidad Autónoma de Madrid

October 11, 2013

[1] based on the original slides of the subject

Index

Index

Programming

What is Programming?

Computer programming (often shortened to programming) is the comprehensive process that leads from an original formulation of a computing problem to executable programs. It involves activities such as:

- The analysis.
- The abstraction.
- The interpretation of problems in algorithms.
- Verification of requirements including its correctness and its resource optimization.
- The coding of the algorithm in a programming language.
- Test, debug, maintain and reuse the code.

Programming

What is Programming?

It can be considered as a balanced measure among:

- An artistic or creative process.
- A craft.
- An engineering discipline.

Software Quality

A program quality is measured by:

- **Reliability**: The results should be corrected. This depends on conceptual correctness of algorithms, and minimization of programming mistakes, such as mistakes in resource management and logic errors (such as division by zero).
- **Robustness**: The program's ability to anticipate problems that are not the logic of the program itself.
- **Usability**: The ease with which a person can use the program for its intended purpose.

Program Quality

- **Portability**: The range of computer hardware and operating system platforms on which the source code of a program can be compiled/interpreted and run.
- **Maintainability**: The ease with which a program can be modified by its present or future developers in order to make improvements or customizations, fix bugs and security holes, or adapt it to new environments.
- **Efficiency**: the amount of system resources a program consumes.

Index

Program

What is a Program?

A computer program, or just a program, is a sequence of instructions, written to perform a specified task with a computer.

Program Design

Design is the process which establishes the requirement, goals, algorithms and different logic and abstraction levels of a program.

- **Abstraction**: It is the conceptual idea of the program and every program part.
- **Modularity**: The different components which conform the program.
- **Flow**: Program behaviour in every action.
- **Data Structure**: The structure of the processed data within the program.
- **Software Architecture**: Final program construction which links all the modules through the flow to establish the general program logic.

Program Design: Programming Paradigm

A programming paradigm is a fundamental style of computer programming, a way of building the structure and elements of computer programs. Some paradigms are:

- **Structure**: This paradigm uses only subroutines and three kind of structures: sequences, selections and repetitions. It tries to avoid unconditional jumps (Goto instruction).
- **Object Oriented**: Concepts are represented as objects which have attributes and methods. The interactions among the objects conform the program.
- **Funcional**: This paradigm uses successive function calls recursively.

Index

Program Design: Example

Definition

Create a program with a client-task list which allows to add or remove information.

- **Abstraction**: The program should:
  - Introduce data about clients and task.
  - Remove data about clients and task.
- A list is used to introduce or remove clients-tasks.

Program Design: Example

Definition

Create a program with a client-task list which allows to add or remove information.

- **Modules**: The different modules are:
  - **List**: Allows to introduce o remove elements.
  - **User**: It communicates with the user.
  - **Data**: Generates the data structure.
  - **Principal**: It manages the modules and the I/O interface.

Program Design: Example

Definition

Create a program with a client-task list which allows to add or remove information.

- **Flow**:
- The users runs the program and he has to choice among:
  - Introduce a new task.
  - Remove a task.
  - Show tasks.
  - Exit.

Program Design: Example

Definition

Create a program with a client-task list which allows to add or remove information.

- **Flow**: Introduce a task.
  - Ask the user the information that he wants to add.
  - Add the information to the list.
  - Return to the beginning.

Program Design: Example

Definition

Create a program with a client-task list which allows to add or remove information.

- **Flow**: Remove a task.
  - Ask the user the information that he wants to remove.
  - Remove the information from the list.
  - Return to the beginning.

Program Design: Example

Definition

Create a program with a client-task list which allows to add or remove information.

- **Flow**: Show task.
  - Show the whole task-list.
  - Print it.
  - Return to the beginning.

Program Design: Example

Definition

Create a program with a client-task list which allows to add or remove information.

- **Flow**: Exit.
  - Free the resources and close the program.

Program Design: Example

Definition

Create a program with a client-task list which allows to add or remove information.

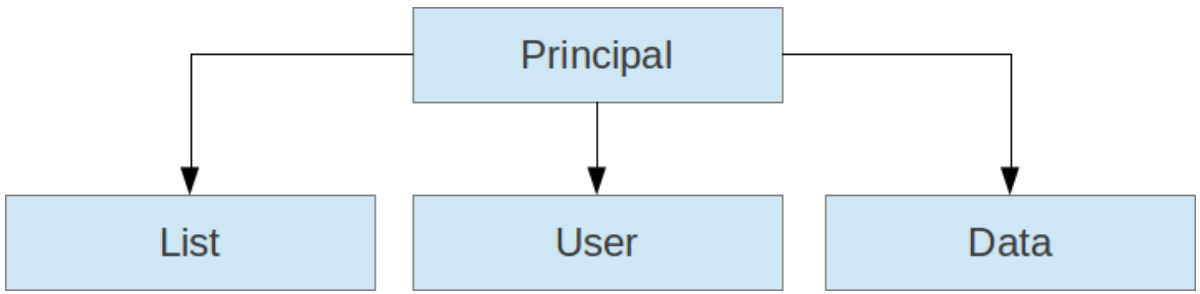
- **Data structure**: The data structure will use the following information:
  - Id: identification number for each element.
  - Client name.
  - Task name.

Program Design: Example

Definition

Create a program with a client-task list which allows to add or remove information.

- **Software Architecture**: Several architectures can be chosen.



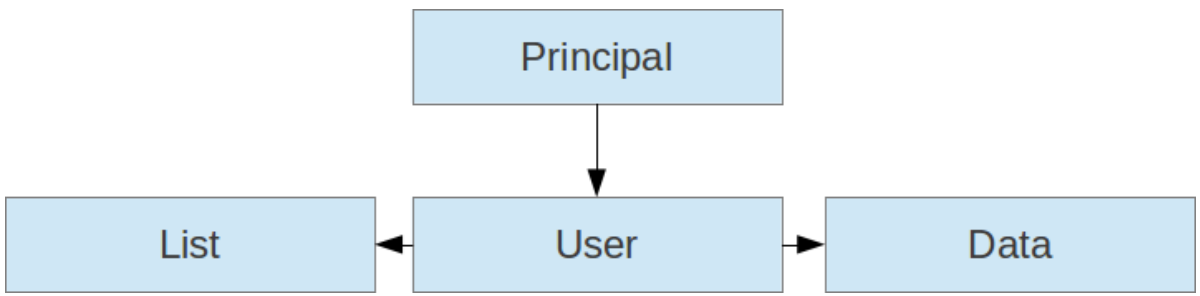
  - List functionalities: Add, remove, ask for data.
  - User functionalities: General Menu, Ask for data (Number or Text), Ask for Questions.

Program Design: Example

Definition

Create a program with a client-task list which allows to add or remove information.

- **Software Architecture**: Several architectures can be chosen.

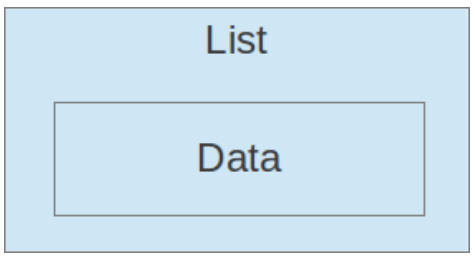


Program Design: Example

Definition

Create a program with a client-task list which allows to add or remove information.

- **Software Architecture**: Several architectures can be chosen.



Index

Coding

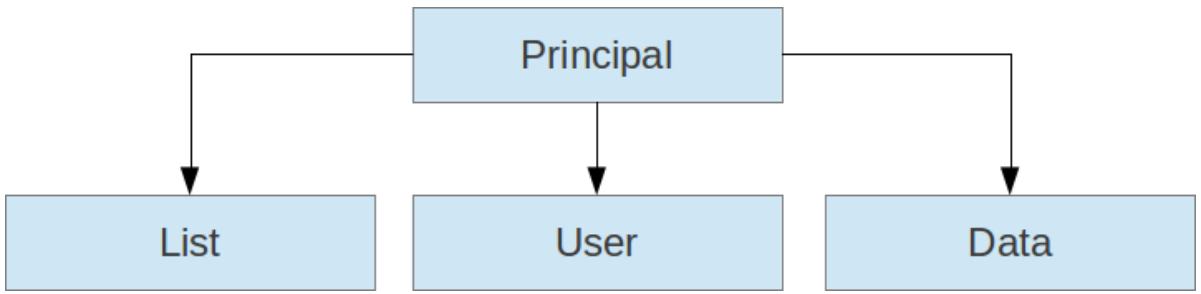
What is coding?

Process which translates from the design to a source code through a Programming language.

Coding

- Algorithm: Sequence of actions used to solve a problem.
- Function: code part which runs a specific task.
- Source code: Written by the developer.
- Libraries: External functions used for a specific task.
- Compiler: Generates the binary files from the source code.
- Runnable: binary file which is runnable.

From Code to Binary

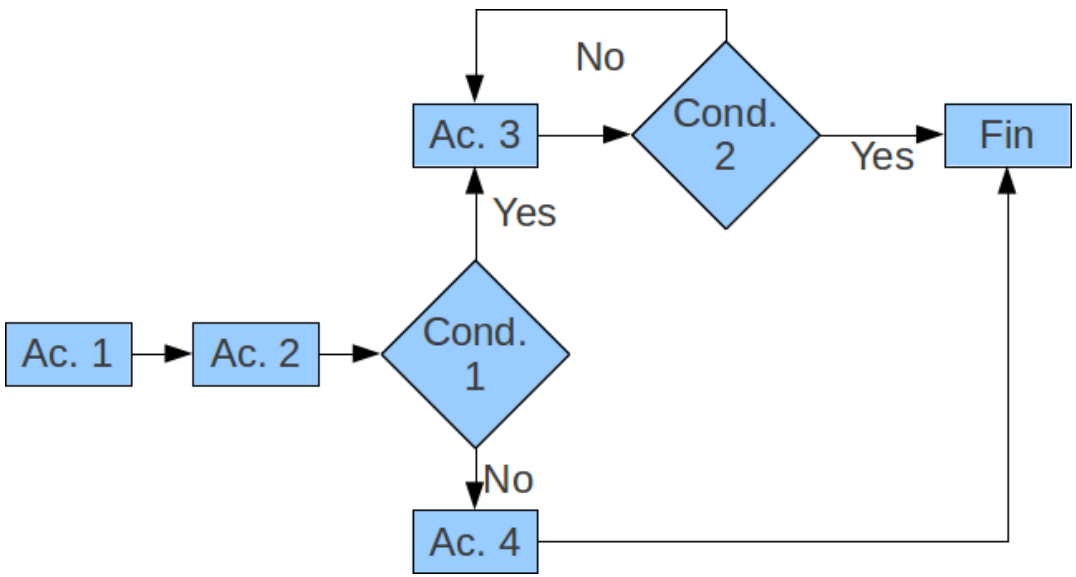


- Principal: Use I/O libraries.
- List and data: libraries.
- User and Principal: Source code.
- Functions: general menu, ask data, ask user...

The paradigms in coding

During this course we will use the paradigm: structure programming. It implies that every program will be written with three kind of instruction:

- Sequential.
- Selective.
- Repetitive.



Index

Introduction to C

C is a high level programming language.

- C advantages:
  - It is small, efficient and stable.
  - There are several programs in C.
  - It is the base for other languages such as C++, Java, AWK or PHP.
- C Disadvantages:
  - It is closed to low level languages.
  - It has several options which makes it harder to learn.

Introduction to C

- Created by Brian Kernighan and Dennis Ritchie in AT& T lab.
- It was created to code UNIX.
- It was firstly described in the first edition of K& R, this description was used as the standard (1978).

Basic Data types in C

- Programs need to use data which shall be stored. This information is saved in variables.
- The type of data stored in the variable is called: data type or type.
- Basic data types in C are:
  - Integer: `int` (5 19 50321)
  - Float or real number: `float` (1.98 3.1415 1.6E19)
  - Character: `char` (a b d)

Basic Data types in C

Definition	Type	Size	From	To
char	Integer	8	-127	127
unsigned char	Integer	8	0	255
short	Integer	16	-32768	32767
unsigned short	Integer	16	0	65535
int	Integer	32	-2147483648	2147483647
unsigned int	Integer	32	0	4294967295
long	Integer	32/64	-2147483648	2147483647
unsigned long	Integer	32/64	0	4294967295
float	Float	32	$3.4 \cdot 10^{-38}$	$3.4 \cdot 10^{38}$
double	Float	64	$1.7 \cdot 10^{-208}$	$1.7 \cdot 10^{208}$

C Variables

- Variables are used to store data values.
- The variable definition needs:
  - The data type, for example, `int`, `char`, `unsigned long`, etc.
  - The variable name.

C Variables

- It is important to consider:
  - Once tha variable is defined, its type can not be change. However, a variable can have several types using `union`.
  - Whether you need other type you can declared other variable with other types or use a `union`.
  - Basic data types store one value. If you want to store several values, you should use arrays or pointers.

C Variables

- The variable name is a referenced to the stored value. It is important to have easy names.
- Digits and letters can be used.
- Upper and lower letters are different. C is "case sensitive". "variable" is different to "Variable".
- Definition examples are:
  - `int var1;`
  - `char miVariable, mivariable, MiVariable;`
  - `unsigned char var2;`

C Variables

- It is important to initialize the variables given them an initial value.
- The assignation is done using the $=$. It can be done when:
  - The variable is declared: `int var1 = 5;`
  - At any time: `var1 = 5;`
- The value of a variable might change during the execution.
- It is possible to define one or more variables:
  - `int var1=5, var2, var3, var4=6;`

C Operators: Arithmetic

- Once the variables have been defined, the user can apply operation such as:

Symbol	Operation
$+$	Sum
$-$	Subtraction
$*$	Multiplication
$/$	Division
%	Module

```
int var1 = 5;
int var2 = var1 - 3;
int var3;
int var4 = 10;
int var5 = var1 * var4;
var3 = var5 % var2;
```

C Operators: Logic

- Logic values: TRUE or FALSE.
- There are not logical variables in C:
  - FALSE is 0
  - TRUE is 1 (or any value $\neq 0$)
- The value of logic operators is defined by their truth table.

C Operators: Logic

Symbol	Operator
==	Equal
!=	Not equal
>	Greater than
>=	Greater or equal than
<	Less than
<=	Less or equal than
&&	AND
||	OR
!	NOT

C Operators: Logic

A	B	A && B	A || B
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	F

C Operators: Others

Use	Description
<var> ++	Use and increment
<var> −−	Use and decrement
++ <var>	Increment and use
−− <var>	Decrement and use

Operator	Description	Example	Equivalent
+=	Sum and assign	`var1 += 3`	`var1 = var1 + 3;`
-=	Subtract and assign	`var1 -= 3`	`var1 = var1 - 3;`
*=	Multiply and assign	`var1 *= 3`	`var1 = var1 * 3;`
/=	Divide y assign	`var1 /= 3`	`var1 = var1 / 3;`

Unidimensional Arrays

- Stores a data set of the same type in consecutive memory positions.
- Their elements are indexed and can be used as variables.
- The first index is 0.
- Definition: `type name[size];`
- Examples:
  - `int a[10];`
  - `int v[4] = {3,5,2,0};`
  - `int w[] = {5,-2,2};`

v[0]	v[1]	v[2]	v[3]
3	5	2	0

Multidimensional Arrays

- It is an array composed by arrays. Similar to a matrix.
- Matrices have two index: rows and columns.
- Each element is indexed by these indexs: `m[2][3]= 5`.
- Definition: `type name[n_rows][n_columns];`
- Examples:
  - `int m[3][4];`

`m[0][0]`	`m[0][1]`	`m[0][2]`	`m[0][3]`
`m[1][0]`	`m[1][1]`	`m[1][2]`	`m[1][3]`
`m[2][0]`	`m[2][1]`	`m[2][2]`	`m[2][3]`

Data Structures

- Matrices group data of the same type.
- If different types are needed the solution are structures.
- A structure is a complex variable. It is formed by simple variables of different types.
- Each variable of the structure has a name.
- Examples:
  - Traffic Radar (car reference - speed)
  - Temperature Sensor (temperature - time)
  - TV (channel - name - frequency)

Data Structures

`struct` is declared as:

Listing 1: struct declaration

```
struct name_struct {
type var_1;
type var_2;
. . .
type var_n;
} ;
```

Data Structures

Example:

Listing 2: struct example

```
struct s_date {
int day;
int month;
int year;
} ;
typedef struct s_date date;
void main() {
date born_date;
//...
}
```

Data Structures

- The structure variables are called members.
- They are accessible through the "." operator.
- They can be managed as any variable.

Data Structures

Example:

Listing 3: struct example

```
struct s_person{
int age;
float weight;
} ;
typedef struct s_person person;
int main()
{
persona p;
p.age = 20;
p.weight = 83.3;
printf("%d", p.age);
printf("%f", p.weight);
}
```

Printf()

- `printf()` is used to print in the terminal.
- `printf("Hello world \n");` prints "Hello world".
- The symbol `\n` is used as new line.
- If you want to print a variable value:

Listing 4: printf

```
printf( String with types, var_1, var_2 );
```

- Where "String with types" is a String which contains elements who represents the different types to print:
  - "%s": for strings.
  - "%i" or "%d": for integer.
  - "%f": for float.
  - "%u": for unsigned int.

Scanf()

- `scanf()` is used to take characters from the standard input.
- It is used as follows:

Listing 5: scanf

```
scanf( String with types, &var_1, &var_2 );
```

- Where "String with types" is a String which contains elements who represents the different types to take:
  - "%s": for strings.
  - "%i" or "%d": for integer.
  - "%f": for float.
  - "%u": for unsigned int.

First Program

- Open a text editor and create a file called `file.c`.
- Write the following code:

Listing 6: hello world

```
#include <stdio.h>
int main(){
printf("Hola mundo\n");
return 0;
}
```

- This program used the standard I/O library: `stdio.h`.
- El programa empieza en `main()`.
- `printf` imprime una cadena de texto.
- `return` finaliza la ejecución.

First Program

- Once you saved the file, open a Terminal and go to the directory which contains it.
- Complite the program using gcc (GNU Compiler Collection):

```
gcc -c file.c
gcc -o program file.o
```

- The first program create the object from `file.c`. The object name is `file.o`. This object is not a program, it is the translation from c code to binary the `file.c`.
- After we link all the object files to generate the final program. In this case there is only one file, however, there are usually many of them. The final program is called `program`.
- If you want to execute it, write:

```
./program
```