Coursework 2: Basic Programming

Héctor Menéndez[1]

AIDA Research Group
Computer Science Department
Universidad Autónoma de Madrid

October 24, 2013

[1] based on the original slides of the subject

Index

Index

Arrays and Loops

- Arrays are usually read and written using loops.

Listing 1: Array examples

```
int main(int argc, char *argv[])
{
    int v[20], i;
    for (i=0; i<20; i++)
    {
        scanf("%d", &v[i]);
    }
    for(i=0; i<20; i++)
    {
        printf("%d", v[i]);
    }
}
```

Arrays and Loops

- Multidimensional Arrays are managed with nested loops.

Listing 2: Array examples

```
int main(int argc, char *argv[])
{
    int m[10][20], i,j ;
    for (i=0; i<10; i++)
    {
        for(i=0; j<20; i++)
        {
            printf("%d", m[i][j]);
        }
    }

}
```

Index

Strings

- Strings are arrays of characters.
- The End Of String is represented by the character '\0'.
- The array size is the string size + 1.
- A string is defined as: `char name[size]`
- Examples:

Listing 3: Strings example

```
int main(int argc, char *argv[])
{
    char cad[5] = {'H', 'o', 'l', 'a', '\0'};
    char cad2[20] = "Hola␣cadena";
    char cad3[ ] = "Asi␣es␣mas␣facil";

}
```

Strings

- Strings use the **string.h** library.
- The most common string functions are:

Function	Description
`char* strcpy(char *cad1, char *cad2)`	Copies cad2 in cad1
`int strcmp(char *cad1, char *cad2)`	Checks whether the two strings are equal
`int strlen(char *cad)`	Measures the length of a string
`char *strtok(char *s1, const char *s2)`	Splits the strings

Strings: Example

Listing 4: Strings example

```
int main(int argc, char *argv[])
{
    char secret="abracadabra";
    char password[50] ;
    scanf("%s", password);
if(strcmp(secret,password) == 0)
        return 1 ;
else
        return 0;
}
```

Index

Enumerates

- Enumerates are used to defined variables types with concrete values.

Listing 5: Enum

```
typedef enum{
    red,
    green,
    blue
}colours;

int main(int argc, char *argv[])
{
    colours colour1 = blue;
    colours colour2 = red;
    printf("%d\n",colour1);
    printf("%d\n",colour2);
}
```

Index

Castings

- A casting use a variable changing its type.
- Example:

Listing 6: Castings

```
int main(int argc, char *argv[])
{
    int a = 5;
    int b = 2;
    float c = 0;
    float d = 0;

    c = a/b;
    d = ((float) a)/((float) b);
    printf("c: %f, d: %f",c,d); //Prints c: 2 d: 2.5
}
```

Castings

- Every character can be represented as an integer.
- The integer value of the character is its ASCII value.
- Casting produces robustness problem in the programs, specially when long variables are casted to small variables.
- Example:

Listing 7: Characters

```
int main(int argc, char *argv[])
{
    char a = 'a';
    printf("char: %c, integer: %d",a,a);
    //Prints char: a integer: 97
}
```

Index

Constants

- Constants are fixed values.
- They can be applied to all the types.
- There are two ways to define them: with `define` and `const`:

Listing 8: Constants

```
#define MAX 100
int main(int argc, char *argv[])
{
    const char* PASS_TAG = "pass";
    printf("MAX number of users: %d",MAX);
    printf("Pass tag: %s",PASS_TAG);
}
```

Index

Functions

- Functions where created to simplify code.
- They are associated to a concrete task.
- The return a value associated with a type (or void).
- They also use parameters.
- `main()` is a special function.
- They have to be define before their use.
- Definition:

Listing 9: Function declaration

```
devolution_type name (parameters)
{
    body;
}
```

Functions: Example

Listing 10: Functions example

```
int sum(int a, int b)
{
    return a+b;
}
int main(int argc, char *argv[])
{
    int i = 4;
    int j = 5;
    int result;
    result  = sum(i,j);
    printf("Sum: %d",result);
}
```

Index

Variables Environment: Local

- An environment for a variable is define by blocks ({ }).
- Functions uses their parameters as variables defined in its environment.
- Local variables: those variables defined inside a block:

Listing 11: Local variables

```
int main(int argc, char *argv[])
{
    int a=2, b=4, c=0;

    printf("%d\n",sum(a,b));
    printf("%d\n",c);
    return 0;
}
```

Variables Environment: Global

- Global variables are define in the highest block level.

Listing 12: Global variables

```
int a,b,c;
void sum()
{
    int c;
    c = a+b;
    printf("%d\n",c);
}
int main(int argc, char *argv[])
{
    a=2;
    b=4;

    sum();
    return 0;
}
```

Index

Function Parameters: Call by value

- Parameters are copies from the original variables.

Listing 13: Call by value

```
int sum (int a, int b)
{
    int c = a + b;
    return c;
}
int main(int argc, char *argv[])
{
    int var1=3 , var2=2, result;
    result = sum (var1, var2) ;
    printf("Result = %d", result);
    return 0 ;
}
```

Function Parameters: Call by reference

- Parameters are references from the original variable. To send the reference we use the character '&'. To use the content of the reference we use '$*$' character.
- The original value can be modified using the reference.

Listing 14: call by reference

```
void sumar ( int a , int b , int *c ) {
    *c = a+b;
}
int main(int argc, char *argv[])
{
    int var1=3 , var2=2, result;
    sum (var1, var2,&result ) ;
    printf("Result = %d", result);
    return 0 ;
}
```

Function Parameters: Main Parameters

- `argc`: used to define the number or arguments used when the program is called.
- `argv`: the parameters as strings.

Listing 15: main example

```
int main(int argc, char *argv[])
{
    int i;
    for (i=0; i<argc; i++)
    {
        printf ("%␣s␣\n", argv[i]);
    }
}
```

Functions: Main Return Value

- `main` output: the return value of main can be looked up using bash.
- Use: `echo $?`

Listing 16: main example

```
int main(int argc, char *argv[])
{
    int i;
    for (i=0; i<argc; i++)
    {
        printf("%s\n", argv[i]);
    }
    return 0;
}
```

- `echo "The output is $?"`