

# Introduction to the C programming language

## Lecture 2

Giuseppe Lipari

`http://retis.sssup.it/~lipari`

Scuola Superiore Sant'Anna – Pisa

November 21, 2008

# Outline

## 1 More on statements

## 2 Arrays

- Exercises
- Strings

## 3 Functions

- Exercises

# Outline

## 1 More on statements

## 2 Arrays

- Exercises
- Strings

## 3 Functions

- Exercises

# do while loop

- An alternative way to write a loop is to use the **do - while** loop

```
do {  
    statement1;  
    statement2;  
    ...  
} while(condition);
```

- The main difference between the **while** and the **do - while** is that
  - in the **while** loop the condition is evaluated before every iteration,
  - in the **do - while** case the condition is evaluated after every iteration
- Hence, with **do - while** the loop is always performed at least once

# Nested loops

- It is possible to define a loop inside another loop. This is very useful in many cases in which we have to iterate on two variables

# Nested loops

- It is possible to define a loop inside another loop. This is very useful in many cases in which we have to iterate on two variables
- What does the following program do?

loops/dloop.c

```
#include <stdio.h>
#include <math.h>

int main()
{
    int i, j;

    printf("%d\n", 2);

    for (i = 3; i <= 100; i = i + 1) {
        for (j = 2; j < i; j = j + 1) {
            if (i % j == 0) break;

            if (j > sqrt(i)) {
                printf("%d\n", i);
                break;
            }
        }
    }

    return 0;
}
```

# Exercises

- 1 Write the equivalence between `while` and `do - while`
- 2 Write the equivalence between `for` and `do - while`
- 3 Write a program that, given two numbers, finds all common factors between them
  - Example 1: 12 and 15, will output 3
  - Example 2: 24 and 12, will output 2, 3, 4, 6

# Reading C programs

- It is very important to be able to learn how to read C programs written by someone else
  - Please, take your time to read programs!
  - You must look at a program as you were the processor: try to “execute a program” on paper, writing down the values of the variables at every step
  - Also, please try to write “clean” programs!
    - so that other programs will find easy to read your own programs



## switch - case

- Sometimes, we have to check several alternatives on the same value; instead of a sequence of if-then-else, we can use a *switch case* statement:

loops/switch.c

```
int main()
{
    int number;

    printf("Enter a number: ");
    scanf("%d", &number);
    switch(number) {
        case 0 :
            printf("None\n");
            break;
        case 1 :
            printf("One\n");
            break;
        case 2 :
            printf("Two\n");
            break;
        case 3 :
        case 4 :
        case 5 :
            printf("Several\n");
            break;
        default :
            printf("Many\n");
            break;
    }
}
```

# Outline

1 More on statements

2 Arrays

- Exercises
- Strings

3 Functions

- Exercises

# Arrays

- Instead of single variables, we can declare arrays of variables of the same type
- They have all the same type and the same name
- They can be addressed by using an index

```
int i;  
int a[10];  
  
a[0] = 10;  
a[1] = 20;  
i = 5;  
a[i] = a[i-1] + a[i+1];
```

- **Very important:** If the array has N elements, index starts at 0, and last element is at N-1
- In the above example, last valid element is a[9]

# Example

arrays/dice.c

```
#include <stdio.h>
#include <stdlib.h>

/* Counts the frequency of occurrence of a number when rolling two dices */
int main()
{
    int i;
    int d1, d2;
    int a[13]; /* uses [2..12] */

    for (i = 2; i <= 12; i = i + 1) a[i] = 0;

    for (i = 0; i < 100; i = i + 1) {
        d1 = rand() % 6 + 1;
        d2 = rand() % 6 + 1;
        a[d1 + d2] = a[d1 + d2] + 1;
    }

    for(i = 2; i <= 12; i = i + 1)
        printf("%d: %d\n", i, a[i]);

    return 0;
}
```

## Quick exercise

- You have no more than 5 minutes to complete this exercise!
- Modify the previous program, so that the user can specify the number of times the two dices will be rolled
- Check that the user do not inserts a negative number in which case you print out an error and exit

# Index range

- What happens if you specify an index outside the array boundaries?

# Index range

- What happens if you specify an index outside the array boundaries?
- The compiler does not complain, but you can get a random run-time error!
- Consider the following program: what will happen?

arrays/outbound.c

```
#include <stdio.h>

int main()
{
    int i;
    int a[10];

    for (i=0; i<15; i++) {
        a[i] = 0;
        printf("a[%d] = %d\n", i, a[i]);
    }

    printf("Initialization completed!\n");
}
```

# Questions

- Index out of bounds is a programming error
  - ① Why the compiler does not complain?
  - ② Why the program does not complain at run-time?
- What is the memory allocation of the program? Where is the array allocated?



# Initialization

- Arrays can be initialized with the following syntax

```
int a[4] = {0, 1, 2, 3};
```

- This syntax is only for static initialization, and cannot be used for assignment

```
int a[4];
```

```
a = {0, 1, 2, 3}; // syntax error!
```

# Matrix

- Two-dimensional arrays (matrixes) can be defined as follows

```
double mat[3][3];  
  
mat[0][2] = 3.5;
```

- It is also possible to define more than 2 dimensions:

```
int cube[4][4][4];
```

- Initialization as follows: `arrays/matrix.c`

```
#include <stdio.h>  
  
int main()  
{  
    int i;  
    double mat[3][3] = {  
        {0, 0, 0},  
        {0, 0, 0},  
        {0, 0, 0}  
    };  
  
    mat[0][2] = 3.5;  
  
    //initialize the whole matrix using single vector indices  
    for (i=0; i<9; i++) {  
        mat[i/3][i%3] = 2.0;  
    }  
    printf("Done\n");  
}
```

# Outline

1 More on statements

2 Arrays

- Exercises
- Strings

3 Functions

- Exercises

# Exercises

- 1 Given 2 arrays of doubles of length 3 that represents vector in a 3-dimensional space, compute the scalar product and the vectorial product
- 2 Given an array of 30 integers, compute max, min and average

# Outline

1 More on statements

2 Arrays

- Exercises
- **Strings**

3 Functions

- Exercises

# Strings

- There is not a specific type for strings in C
- A string is a sequence of char terminated by value 0
- To store strings, it is possible to use arrays of chars

```
char name[20];
```

- Initialization:

```
char name[20] = "Lipari";
```

- But again, this syntax is not valid for assignments!
- In memory:

	L	i	p	a	r	i	0			
name	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

# String length

- **IMPORTANT:** if you need a string with 10 characters, you must declare an array of 11 characters! (one extra to store the final 0)
- Computing string length

```
char s[20];  
...  
// how many valid characters in s?  
int i;  
for (i=0; i<20; i++) if (s[i] == 0) break;  
  
if (i<20) printf("String is %d characters long\n", i);  
else printf("String is not valid!\n");
```

- What is in a string? `strings/contents.c`

```
#include <stdio.h>  
  
int main()  
{  
    int i;  
    char str[20] = "donald duck";  
  
    for (i=0; i<20; i++)  
        printf("%d ", str[i]);  
    printf("\n");  
}
```

# String manipulation functions

`int strcpy(char s1[ ], char s2[]);` copies string s2 into string s1  
`int strcmp(char s1[ ], char s2[]);` compare strings alphabetically  
`int strcat(char s1[ ], char s2[]);` append s2 to s1  
`int strlen(char s[ ]);` computes string length  
`printf("%s", str);` prints string on screen



# Safe versions

- Previous functions are not safe: if the string is not well terminated, anything can happen
- There are safe versions of each:
  - `int strncpy(char s1[ ], char s2[], int n);` copies at most n characters
  - `int strncat(char s1[ ], char s2[], int n);` appends at most n characters
  - `int strncmp(char s1[ ], char s2[], int n);` compares at most n characters

# Outline

1 More on statements

2 Arrays

- Exercises
- Strings

3 Functions

- Exercises

# Function definition and declaration

- A function is defined by:
  - a unique name
  - a return value
  - a list of arguments (also called parameters)
  - a body enclosed in curly braces

```
/* returns the power of x to y */  
double power(double x, int y)  
{  
    int i;  
    double result = 1;  
  
    for (i=0; i < y; i++)  
        result = result * x;  
  
    return result;  
}
```

# Function call

functions/power.c

```
int main()
{
    double myx;
    int myy;
    double res;

    printf("Enter x and y\n");
    printf("x? ");
    scanf("%lg", &myx);
    printf("y? ");
    scanf("%d", &myy);

    res = power(myx, myy);

    printf("x^y = %lgt\n", res);
}
```

# Parameters

- Modifications on local parameters have no effect on the caller

```
int multbytwo(int x)
{
    x = x * 2;
    return x;
}

int main()
{
    ...
    i = 5;
    res = multbytwo(i);
    /* how much is i here? */
    ...
}
```

- x is just a *copy* of i
- modifying x modifies the copy not the original value

# Array parameters

- We say that parameters are passed *by value*
  - every time we call the function, a copy is made
- There is only one exception to this rule: when we pass arrays!
  - The array is not copied, so modification to the local parameter are immediately reflected to the original variable

# Array parameters

functions/swap.c

```
#include <stdio.h>

void swap (int a[])
{
    int tmp;
    tmp = a[0];
    a[0] = a[1];
    a[1] = tmp;
    return;
}

int main()
{
    int my[2] = {1,5}
    printf ("before swap: %d %d",
           my[0], my[1]);

    swap(my);

    printf ("after swap: %d %d",
           my[0], my[1]);
}
```

- The array is not copied
- modification on array a are reflected in modification on array my
  - (this can be understood better when we study pointers)
- Notice also:
  - the swap function does not need to return anything: so the return type is void
  - the array my is initialized when it is declared

# Outline

## 1 More on statements

## 2 Arrays

- Exercises
- Strings

## 3 Functions

- Exercises



# Exercises

- 1 Write a function that, given a string, returns its length
- 2 Write a function that, given two strings `s1` and `s2`, returns 1 if `s2` is contained in `s1`
- 3 Write a function that given a string, substitutes all lower case characters to upper case