Introduction to Systems Programming

Antonio Carzaniga

Faculty of Informatics
Università della Svizzera italiana

September 19, 2022

General Information

- ***https://www.inf.usi.ch/carzaniga/edu/sysprog/***

- on iCorsi: ***INF.B.SA22-23.17***

General Information

- ***https://www.inf.usi.ch/carzaniga/edu/sysprog/***
- on iCorsi: ***INF.B.SA22-23.17***
- Announcements
  - ***https://www.inf.usi.ch/carzaniga/edu/sysprog/***
  - or through iCorsi

  you are responsible for reading the announcements page or reading the announcements sent through iCorsi

General Information

- ***https://www.inf.usi.ch/carzaniga/edu/sysprog/***

- on iCorsi: ***INF.B.SA22-23.17***

- Announcements
  - ***https://www.inf.usi.ch/carzaniga/edu/sysprog/***
  - or through iCorsi

  you are responsible for reading the announcements page or reading the announcements sent through iCorsi

- Office hours
  - Antonio Carzaniga: *by appointment*
  - Shamiek Mangipudi: *by appointment*
  - Arnaud Fauconnet: *by appointment*

Goals and Structure

- Focus: ***concrete and practical*** systems programming
  - still with good software engineering practices

Goals and Structure

- Focus: ***concrete and practical*** systems programming
  - still with good software engineering practices

- Structure: ***reading/lecture*** + ***in-class exercises*** + ***homework***

Goals and Structure

- Focus: ***concrete and practical*** systems programming
  - still with good software engineering practices
- Structure: ***reading/lecture*** + ***in-class exercises*** + ***homework***
- *Lectures*
  - *preliminary reading* plus *interactive lectures*
  - in-class exercises
  - so, you should have your computer handy (and charged)

Goals and Structure

- Focus: ***concrete and practical*** systems programming
  - still with good software engineering practices
- Structure: ***reading/lecture*** + ***in-class exercises*** + ***homework***
- *Lectures*
  - *preliminary reading* plus *interactive lectures*
  - in-class exercises
  - so, you should have your computer handy (and charged)
- *Homework*
  - a programming assignment *every week*
  - a few assignments will be graded (we'll tell you which ones)
  - all assignments will be discussed in class

How to Learn Systems Programming

How to Learn Systems Programming

1. Solve a programming problem

2. If you are stuck, ask somebody to help you—ask me (Antonio) to help you!
   . . . but *do not simply copy code!*

3. When you're done—when your own solution is *complete*—analyze other solutions, such as Antonio's solutions presented in class

4. Go to step 1

Evaluation

Evaluation

- +30% homework: programming assignments
  - grades added together, thus resulting in a weighted average
- +30% midterm exam
  - in-class programming using your computer
- +40% final exam
  - in-class programming using your computer
- ±10% instructor’s discretionary evaluation
  - participation
  - extra credits
  - trajectory
  - . . .

Evaluation

- +30% homework: programming assignments
  - grades added together, thus resulting in a weighted average
- +30% midterm exam
  - in-class programming using your computer
- +40% final exam
  - in-class programming using your computer
- ±10% instructor’s discretionary evaluation
  - participation
  - extra credits
  - trajectory
  - . . .
- −100% plagiarism penalties

Plagiarism

Plagiarism

A student should never take someone else’s material and present it as his or her own. Doing so means committing plagiarism.

Plagiarism

A student should never take someone else’s material and present it as his or her own. Doing so means committing plagiarism.

- You know what I mean. . .

Plagiarism

A student should never take someone else's material and present it as his or her own. Doing so means committing plagiarism.

- You know what I mean…
- Committing plagiarism on an assignment or an exam will result in
  - failing that assignment or that exam
  - loosing one or more points *in the final note!*
- Penalties may be escalated…

Deadlines

Deadlines

Deadlines are firm.

Deadlines

Deadlines are firm.

- You know what I mean. . .
- Usual three-days-and-you’re-out rule applies here. . .

Now on to ***Systems Programming!***

What is Systems Programming?

What is Systems Programming?

- Interfacing with a “system” (as opposed to a user)
  - rigid interfaces
  - complex interfaces

What is Systems Programming?

- Interfacing with a “system” (as opposed to a user)
  - rigid interfaces
  - complex interfaces
- Engineering for a non trivial platform
  - non-trivial performance profiles
  - going beyond algorithmic complexity

The Language(s) of Systems Programming

The Language(s) of Systems Programming

- Mostly C, and a bit of C++

The Language(s) of Systems Programming

- Mostly C, and a bit of C++

- A lot of software is written in C or C++
  - the vast majority of the programs running on your computer (now!)
  - including the operating system
  - a lot more *new* software will be written in C/C++
  - usually the most crucial components of large systems

The Language(s) of Systems Programming

- Mostly C, and a bit of C++
- A lot of software is written in C or C++
  - the vast majority of the programs running on your computer (now!)
  - including the operating system
  - a lot more *new* software will be written in C/C++
  - usually the most crucial components of large systems
- Available on virtually every computer platform
  - from embedded controllers to supercomputers

The Language(s) of Systems Programming

- Mostly C, and a bit of C++
- A lot of software is written in C or C++
  - the vast majority of the programs running on your computer (now!)
  - including the operating system
  - a lot more *new* software will be written in C/C++
  - usually the most crucial components of large systems
- Available on virtually every computer platform
  - from embedded controllers to supercomputers
- System programming
  - “low-level” programming (e.g., a device driver)
  - “high-level” programming (e.g., the Firefox web browser)

The Language(s) of Systems Programming

- Mostly C, and a bit of C++
- A lot of software is written in C or C++
  - the vast majority of the programs running on your computer (now!)
  - including the operating system
  - a lot more *new* software will be written in C/C++
  - usually the most crucial components of large systems
- Available on virtually every computer platform
  - from embedded controllers to supercomputers
- System programming
  - "low-level" programming (e.g., a device driver)
  - "high-level" programming (e.g., the Firefox web browser)
- Relatively simple (C) but still powerful language
  - C++ is definitely not that simple
  - like any serious tool, C and C++ have hidden complexities. . .

Getting Started: One, Two, Three!

Getting Started: One, Two, Three!

1. Edit the program *ciao.c*

```
#include <stdio.h>

int main () {
    print(”Ciao!\n”);
}
```

Getting Started: One, Two, Three!

1. Edit the program *ciao.c*

```
#include <stdio.h>

int main () {
   print(”Ciao!\n”);
}
```

2. Compile the program (i.e., run the compiler)

```
$ cc ciao.c -o ciao
```

Getting Started: One, Two, Three!

1. Edit the program *ciao.c*

```
#include <stdio.h>

int main () {
   print(”Ciao!\n”);
}
```

2. Compile the program (i.e., run the compiler)

```
$ cc ciao.c -o ciao
```

3. Run the program

```
$ ./ciao
```

Getting Started with C++

Getting Started with C++

1. Edit the program *ciao2.cc*

```
#include <iostream>

int main () {
    std::cout << "Ciao!\n";
}
```

Getting Started with C++

1. Edit the program *ciao2.cc*

```
#include <iostream>

int main () {
    std::cout << "Ciao!\n";
}
```

2. Compile the program (i.e., run the compiler)

```
$ c++ ciao2.cc -o ciao2
```

Getting Started with C++

1. Edit the program *ciao2.cc*

```
#include <iostream>

int main () {
    std::cout << "Ciao!\n";
}
```

2. Compile the program (i.e., run the compiler)

```
$ c++ ciao2.cc -o ciao2
```

3. Run the program

```
$ ./ciao2
```

Getting Started with *Make*

Getting Started with *Make*

1. Edit the program *ciao3.cc*

```
#include <iostream>
int main() {
    std::cout << ”I said Ciao already!\n”;
}
```

Getting Started with *Make*

1. Edit the program *ciao3.cc*

```
#include <iostream>
int main() {
    std::cout << ”I said Ciao already!\n”;
}
```

2. Compile the program using *make*

```
$ make ciao3
```

Getting Started with *Make*

1. Edit the program *ciao3.cc*

```
#include <iostream>
int main() {
   std::cout << ”I said Ciao already!\n”;
}
```

2. Compile the program using *make*

```
$ make ciao3
```

3. Run the program

```
$ ./ciao3
```

Errors

Errors

Try compiling the program:

```
#include <iostream>

int main() {
    cout << ”I said Ciao already!\n”;
}
```

Errors

Try compiling the program:

```
#include <iostream>

int main() {
    cout << "I said Ciao already!\n”;
}
```

You should get some errors:

```
$ g++ errors.cc -o errors
errors.cc: In function ‘int main()’:
errors.cc:4:5: error: ‘cout’ was not declared in this scope
...
```

Printing

The function you will use to print data in C is `printf`:

```c
#include <stdio.h>

int main() {
    printf("My name is %s.\nI was %d in the year 2000.\n”,
           "Antonio”, 2000 - 1969);
}
```

Printing

The function you will use to print data in C is `printf`:

```
#include <stdio.h>

int main() {
   printf("My name is %s.\nI was %d in the year 2000.\n",
         "Antonio", 2000 - 1969);
}
```

The first argument is a ***format string*** that includes ***conversion specifications***, begining with a `%` sign, that tell printf how to interpret its other arguments:

`%d` prints an integer in decimal notation
`%c` prints an integer as a character
`%g` prints a `float` in decimal notation
… *see the documentation of printf()*

Printing in C++

Printing is quite different (simpler?) in C++:

```
#include <iostream>

int main() {
    std::cout
        << "My name is ” << "Antonio”
        << ".\nI was ” << 2000 - 1969
        << ” in the year 2000.\n”;
}
```

Digression: How does this really work?

Minimal (One-Byte) I/O

- `getchar()` reads the next character (byte) from the “standard input”
  - returns an `int` value
  - returns `EOF` at the end of file

Minimal (One-Byte) I/O

- `getchar()` reads the next character (byte) from the “standard input”
  - returns an `int` value
  - returns `EOF` at the end of file

- **Exercise:** write a program that counts and prints the number of characters (bytes) in its standard input stream

Minimal (One-Byte) I/O

- `getchar()` reads the next character (byte) from the “standard input”
  - returns an `int` value
  - returns `EOF` at the end of file

- **Exercise:** write a program that counts and prints the number of characters (bytes) in its standard input stream

```
#include <stdio.h>

int main() {
    int i = 0;
    while (getchar() != EOF)
        ++i;
    printf ("%d characters\n", i);
    return 0;
}
```

Examples. . .

Minimal (One-Byte) I/O

`putchar(int c)` writes one byte to the “standard output”

Minimal (One-Byte) I/O

`putchar(int c)` writes one byte to the “standard output”

- **Exercise:** what does this program do?

```c
#include <stdio.h>
#include <limits.h>

int main() {
    int c;
    while ((c = getchar()) != EOF) {
        c += 3;
        if (c > UCHAR_MAX)
            c = UCHAR_MIN + (c - UCHAR_MAX);
        putchar(c);
    }
}
```

Minimal (One-Byte) I/O

`putchar(int c)` writes one byte to the “standard output”

- **Exercise:** what does this program do?

```c
#include <stdio.h>
#include <limits.h>

int main() {
    int c;
    while ((c = getchar()) != EOF) {
        c += 3;
        if (c > UCHAR_MAX)
            c = UCHAR_MIN + (c - UCHAR_MAX);
        putchar(c);
    }
}
```

- **Exercise:** write a program that inverts the transformation of the program above

Characters vs. Octets

Run this program:

```
#include <stdio.h>

int main () {
    putchar(67);
    putchar(105);
    putchar(97);
    putchar(111);
    putchar(33);
    putchar(10);
}
```

Characters vs. Octets

Now run this other program:

```c
#include <stdio.h>

int main () {
    putchar(240);
    putchar(159);
    putchar(153);
    putchar(130);
    putchar(10);
}
```

Control Structures

C and C++ have the usual control structures:

- `for`
- `while`
- `do...while`
- `switch`
- `if...else...`
- `break`
- `continue`
- `return`

```
int f(int n) {
    int p, pp, r;
    switch(n) {
    case 0:
    case 1: return n;
    default:
        p = 1;
        pp = 0;
        do {
            r = p + pp;
            pp = p;
            p = r;
        } while (--n > 1);
        return r;
    }
}
```

Control Structures: Exercise 1

- Write a program called `diamond.c` that, given a number *n*, prints (on the terminal) an $n \times n$ diamond like this one (6×6):

```
     #
    ###
   #####
  #######
 #########
###########
 #########
  #######
   #####
    ###
     #
```

Control Structures: Exercise 2

- Rewrite without using the `switch` statement

```
int main () {
    int c;
    while ((c = getchar()) != EOF) {
        switch (c) {
        case ' ': putchar('\n'); break;
        case '\n': putchar('\n'); putchar('\n'); break;
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u': putchar(c); putchar('s');
        default: putchar(c);
        }
    }
}
```

Control Structures: Exercise 3

- Write a program that *reverts* this input/output transformation:

```c
int main () {
    int c;
    while ((c = getchar()) != EOF) {
        switch (c) {
        case ' ': putchar('\n'); break;
        case '\n': putchar('\n'); putchar('\n'); break;
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u': putchar(c); putchar('s');
        default: putchar(c);
        }
    }
}
```

Homework Assignment: wordcount

- Write a program called *wordcount* that counts the words in the standard input. A *word* is a sequence of one or more characters delimited by white space.
  - the output should be the same as the command:

```
$ wc -w
```