

EL2310 – Scientific Programming

Lecture 11: Introduction to C++

Hakan Karaoguz (hkarao@kth.se)

Credits: Ramviyas Parasuraman, Andrej Pronobis (U.Washington), Florian Pokorny (UC Berkeley)

KTH - Royal Institute of Technology

Overview

Lecture 11: Introduction to C++

- Wrap Up

- Introduction to C++

- Differences between C and C++

- Printing and User Input

- Namespaces

Wrap Up

Differences between C and C++

Printing and User Input

Namespaces

- ▶ External Input
- ▶ Reading/writing files
- ▶ Bitwise operations

- ▶ `int main(int argc, char **argv)`
- ▶ Arguments are separated by spaces: `./test 12 john`
- ▶ **argc**: argument count **argv**: argument vector
- ▶ If nothing is inputted, then `argc` is 1 and `argv[0]` is the name of the executable
- ▶ `atoi` and `atof` are useful to get number from char arrays

- ▶ `int main(int argc, char **argv)`
- ▶ Arguments are separated by spaces: `./test 12 john`
- ▶ **argc**: argument count **argv**: argument vector
- ▶ If nothing is inputted, then `argc` is 1 and `argv[0]` is the name of the executable
- ▶ `atoi` and `atof` are useful to get number from char arrays

Reading from the keyboard

- ▶ Can use `char getchar();` to get a single character
- ▶ For more more complex input try `scanf(...)` which is the “dual” of `printf(...)`
- ▶ The arguments for `scanf` the same as for `printf` except that it wants pointers to where to put the data

Opening/closing a file

- ▶ `FILE *fopen(char *path, char *mode);`
- ▶ mode is “r”: read, “w”: write, “a”:append, ...
- ▶ On success returns pointer to file descriptor, else NULL
- ▶ `fclose(FILE*);`

- Ex: double x=1, y=2, theta=0.5;

$$\text{if (fd) \{}$$

```
x, y, theta);
```

$$\}$$

Reading from a file

- ▶ Read from the file with for example

- ▶ `fscanf(FILE*, ...);`

- ▶ **Ex:** `double x, y, theta;`

```
FILE *fd = NULL;
```

```
fd = fopen("test.txt", "r");
```

```
fscanf(fd, "Robot pose is %lf %lf %lf\n",
```

```
&x, &y, &theta);
```

```
fclose(fd);
```

- ▶ Use `EOF` keyword to check whether you have reached to the End Of File

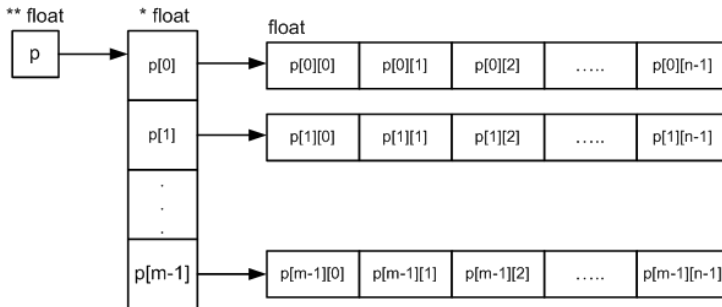
Bitwise operations

- ▶ When programming at low level, bitwise operations are common (Embedded programming using Microcontrollers, Arduinos etc.)
- ▶ Memory efficiency. In C all the variables are multiples of bytes (1 byte = 8 bits = [0-255]) but for flags you just need 1 bit.
- ▶ Typical construction, use *bitmask*
- ▶ The operators do not change the actual value of the variable, they create a temporary variable.

Discussion on C assignment

- ▶ Given example function declarations are not mandatory, you can change them
- ▶ The file reading part can be solved like this:
 - ▷ Write a `determineDataSize` function that determines the number of rows and cols in the *data.txt* file
 - ▷ The return type of this function can be a `struct` with members `rows` and `cols`
 - ▷ Allocate your double pointer array (`double**`) `data` based on determined rows and cols in the main function
 - ▷ Actually read the file contents using the initialized `data` variable
- ▶ Some concepts are a bit challenging?
- ▶ Do you need more time to understand what you are doing?

The concept of double pointer



DYNAMIC ARRAY (2D) USING POINTER TO ARRAY OF POINTERS

- Pointer 'p' points to an array of pointers to float
- Each element of this array points to a block of memory sufficient to store one row
- The way memory is allocated, it is possible that the memory allocated to the different rows may not be contiguous

Lecture 11: Introduction to C++

Wrap Up

Introduction to C++

Differences between C and C++

Printing and User Input

Namespaces

Course so far

- ▶ MATLAB: Using program to perform some tasks
- ▶ C: Learning how to program

Rest of the course

- ▶ C++
 - ▷ Writing extendable (modular) programs in C++
 - ▷ Object Oriented Programming
 - ▷ Using other people's code
 - ▷ Modifying/Extending other people's code
 - ▷ Writing re-usable code

Today

- ▶ Basics in C++
- ▶ Introduction to OOP

What is C++?

- ▶ Developed by Bjarne Stroustrup starting from 1979 at Bell Labs
- ▶ Adds object oriented features (e.g. classes) to C
- ▶ Initially named: C with Classes; then renamed to "C++" (guess why?)
- ▶ Influenced many other languages: C#, Java
- ▶ The C++ standard library incorporates:
 - ▷ The C standard library with small modifications
 - ▷ STL (Standard Template Library)
- ▶ Constantly developed: C++11 (2011), C++14 (2014)

Lecture 11: Introduction to C++

Wrap Up

Introduction to C++

Differences between C and C++

Printing and User Input

Namespaces

C++ subsumes C

- ▶ You can use all you learned in C in C++ as well
- ▶ Some constructs/syntax have a C++ version

File naming conventions

- ▶ We named files in C as `.c` (source) and `.h` (header)
- ▶ In C++, the ending is typically `.cc` or `.cpp` for source files and `.h`, `.hh` or `.hpp` for header files
- ▶ In this course we will use `.cpp` and `.h`

C++ Compiler

- ▶ `g++` and `gcc`: `g++` is specific to C++ by (auto) linking to std C++ libraries whereas `gcc` decides based on the file extension (`.c/.cpp`)
- ▶ Usage and command line options for `g++` are the same as for `gcc`
- ▶ Make sure you know how to use `make` for this part of the course!

Comments in C++

- ▶ Multi-line comments as in C, i.e. `/* ... */`
- ▶ Single-line comments using `//`
- ▶ Example:

```
int main() {  
    // This is a single line comment  
    /* This comment extends to  
multiple lines */    ...  
}
```

Basic data types

- ▶ All data types from C can be used. Plus some more, e.g.
- ▶ `bool`: **boolean value** `true/false`
- ▶ `string`: “real” string (use `#include <string>`)

Declaration of variables

- ▶ You no longer need to declare the variable at the beginning of the function (scope), as was the case for pre C99
- ▶ Rule of thumb: declare variables close to where they're used.
- ▶ For instance:

```
for (int i=0; i<N; i++) {...}
```

`i` only defined within loop

- ▶ Use specific names for counters, e.g. `i`, `j`, `k`, ...

Lecture 11: Introduction to C++

Wrap Up

Introduction to C++

Differences between C and C++

Printing and User Input

Namespaces

Printing to Screen

- ▶ In C++ we use *streams* for input and output using the `<iostream>` library
- ▶ Output is handled with the stream `cout` and `cerr`, using the `<<` operator
Ex: `std::cout << "Hello world";`
- ▶ To add a line feed use the “\n” as in C or the special `endl`
`std::cout << "Hello world\n" ;`
`std::cout << "Hello world" << std::endl;`
- ▶ All basic data types have the ability to add themselves to a stream for printing

Printing to screen cont'd

- ▶ You can mix data types easily
- ▶ In C: `printf("The value is %d\n", value);`
- ▶ In C++: `std::cout << "The value is " << value << endl;`
- ▶ The stream `cerr` is the error stream
- ▶ Compare `stdout` and `stderr` in C

Getting input from the user

- ▶ streams is also used to get input from console

- ▶ Use the `cin` stream

- ▶ Ex:

```
int value;  
std::cout<<"Please enter an integer value:  ";  
std::cin >> value;
```

- ▶ When reading multiple inputs, they are separated by spaces

- ▶ Ex: `std::cin >> a >> b >> c;`

Reading strings

- ▶ If you want to read an entire line into a string, use `getline`

- ▶ Ex:

```
string line;  
getline(cin, line);  
cout << "The input was " << line << endl;
```

Hello KTH in C++ vs C

```
#include <iostream>
int main ()
{
    std::cout << "Hello
KTH!"<<std::endl;
    return 0;
}
```

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

Hello KTH in C++ vs C

- ▶ `<iostream>` replaced `<stdio.h>`
- ▶ Standard C++ header files are included without the suffix (no `.h` at the end)
- ▶ Here the `std` namespace is used, where `cout` is found.
More to come about namespaces!!

Task 1

- ▶ Please go to web address *cpp.sh* or *<https://www.codechef.com/ide>*
- ▶ Write a C++ program that asks the name and age of a person
- ▶ It should then print this info on the screen

Lecture 11: Introduction to C++

Wrap Up

Introduction to C++

Differences between C and C++

Printing and User Input

Namespaces

Namespaces

- ▶ In C all functions share a common `namespace`
 - ▷ This means that there can only be one function for each function name
- ▶ In C++ functions can be placed in specific namespaces
- ▶ Syntax:

```
namespace NamespaceName {  
    void fcn(); ...  
}
```

- ▶ To access a function `fcn` in namespace `A`,
`A::fcn`
- ▶ This way you can have more than one function with the same name but in different namespaces

- ▶ To access a function `fcn` in namespace `A`,
`A::fcn`
- ▶ This way you can have more than one function with the same name but in different namespaces

- ▶ Specifying the namespace all the time == lose time in typing
`std::cout << "Who likes typing?" << std::endl;`
- ▶ Solution: extending a specific namespace in a program,
- ▶ E.g.

```
using namespace std  
cout << "OK" << endl;  
cout << "Now it feels much better!" << endl;
```
- ▶ But avoid using this in header files

