Coşkun Şahin

coskun.sahin1@ucalgary.ca
ICT 526

C/C++ Review

CPSC 457, Winter 2019
Department of Computer Science
University of Calgary

Environment

- Fedora 28
- **gcc 8.2.1**
- **g++ 8.2.1**
- Your favorite text editor/IDE
  - Vi/Vim
  - GNU Emacs
  - geany
  - gedit
  - Atom
- SSH
  - username@linux.cpsc.ucalgary.ca
  - PuTTY for Windows
- Sample codes are at https://github.com/coskunsahin1/CPSC457

Why should I learn C/C++ now?

- High level as well as low level language
  - From writing OS kernel to writing an application program
- Gives more control over low level mechanisms
  - Memory management, memory location management, mixing assembly code, device management, direct access to OS primitives
- Performance is *sometimes* better. Execution is also more predictable (no random garbage collection).
- Most OS code (Linux, BSD) is written in C (so is Sun JVM)
- But has some downsides against Java
  - Requires careful memory management.
  - You may have to write more lines of code.
  - More room for mistakes (memory leaks, initialization errors)
  - Code is not (directly) portable.

Outline

- **Compiling and executing a C/C++ program**
- Data types
- File I/O
- C++ classes and objects (only for C++)
- Pointers
- Libraries

"Hello World!"

Step 1: Write code using your favorite editor

C program: hello.c

```
#include <stdio.h>

int main(int argc, char * argv[])
{
    printf("Hello World!\n");
    return 0;
}
```

C++ program: hello.cpp

```
#include <iostream>

using namespace std;

int main(int argc, char * argv[])
{
    cout << "Hello World!" << endl;
    return 0;
}
```

"Hello World!"

Step 2: Compile

1. Make sure **gcc** (for C) and **g++** (for C++) are installed.
2. Change current directory to where the C++ source file was saved
3. Compile your program

   ```
   g++ hello.cpp -o hello
   gcc hello.c -o hello
   ```

   (By default, the executable program is under the name **a.out**. The **-o** option allow you to change the name.)

gcc and g++

- Common parameters for **gcc** and **g++**

-o *file*	output file for object or executable
-Wall	all warnings – use always!
-c	compile single module (non-main)
-g	insert debugging code (gdb)
-l	library

http://www.cs.columbia.edu/~hgs/teaching/ap/slides/CforJavaProgrammers.ppt

Command-Line Arguments

Number of argumentsThe array of arguments

```
int main(int argc, char * argv[])
```

Return value of the program: 0 = success, others = some error
Can also be declared as void, *i.e.*, no return value

- Try to execute the sample **`args.c`** or **`args.cpp`** programs

-
  ```
  $ g++ args.cpp -o args
  $ ./args
  $ ./args 1 2 "hello"
  ```

Outline

- Compiling and executing a C++ program
- **Data types**
- File I/O
- C++ classes and objects (only for C++)
- Pointers
- Libraries

Primitive Data Types

- `bool` (just in C++)
- `char`: a single character
- `short, int, long, long long`: integers
- `float, double`: floating point numbers
- You can also define your own types using `typedef`
  - `typedef unsigned char byte`
- basic_io.c / basic_io.cpp
- Enumerated types
  - **enum cardsuit { CLUBS = 1, DIAMONDS = 2, HEARTS = 3, SPADES = 4 };**

Size and Range

Type	**Bytes**	**Range**
char	1	-128 … 127
short	2	-65536…65535
int, long	4	2^{32} or -2,147,483,648 to 2,147,483,647
long long	8	2^{64}
float	4	3.4E+/-38 (7 digits)
double	8	1.7E+/-308 (15 digits)

http://www.cs.columbia.edu/~hgs/teaching/ap/slides/CforJavaProgrammers.ppt

Array (`array.c`)

- Declaration: **`int array[size]`**
  - The size must be provided for static allocation
    - More on dynamic allocation later

- When passing an array to a function, typically you have to pass the array size as a separate argument as well.
  - C/C++ arrays have no length attribute
  - `foo(array, `**`size`**`);`

- You have to take care of array bounds yourself
  - `int input[10];`
  - `input[`**`10`**`] = 20;`
  - `input[`**`-1`**`] = 5;`

All "work", but can cause serious and unexpected issues, or even crash your program.

Structures (struct)

- C/C++ **`struct`** is a way to **logically group related types**.
- Is very similar to (but not same as) C++/java **classes**
  - **`struct`** is a class without methods
- Accessed in **struct.field** manner.
  - In C/C++, **`struct`** fields are public by default
  - C does not have any OO features like encapsulation.
-
```
struct student
{
      int studentID;
      float mark;
};
```

Strings

- In C, string is an array of `char` ended with "\0" (a null terminator)
  - e.g., **`char str[6];`**
    ```
    str = "hello\0";
    printf ("%s\n", str);
    ```

- In C++, there is a string library that provides a string class much similar to the one in java
  - e.g., **`string str = "hello";`**
    ```
    cout << str << endl;
    ```

Outline

- Compiling and executing a C++ program
- Data types
- **File I/O**
- C++ classes and objects (only for C++)
- Pointers
- Libraries

File I/O in C (`fileIO.c`)

- Open a file: `FILE * fd = fopen (filename, mode);`
  - e.g., `FILE * fd = fopen (“~/HW/input.txt", "r");`

r - open for reading
w - open for writing (file need not exist)
a - open for appending (file need not exist)
r+ - open for reading and writing, start at beginning
w+ - open for reading and writing (overwrite file)
a+ - open for reading and writing (append if file exists)

- Close a file: `fclose(fd);`
- Writing to a file: `fprintf(fd, "Hello\n");`
- Reading from a file: `fscanf(fd, "%s", str);`
  - Return a special value `EOF` when reading the end of a file
- It is also possible to read and write character by character from and to a file using `fgetc()` and `fputc()`, respectively.

File I/O in C++ (`fileIO.cpp`)

- `#include <fstream>`

- Open a file for writing: `ofstream outfile;`
  `outfile.open("myfile.txt");`

- Open a file for reading: `ifstream infile;`
  `infile.open("myfile.txt");`

- Open a file for both writing and reading:
  `fstream file;`
  `file.open("myfile.txt");`

- Close a file: `outfile.close(); infile.close(); file.close()`

- Writing to a file: `outfile << "hello\n";`

- Reading from a file: `getline(infile, str);`

- Check for end of file: `infile.eof();` returns bool.

Exercise

- Write a C++ program and save it under the file name readWords.cpp. The program outputs every word in a newline to the standard output.

Outline

- Compiling and executing a C++ program
- Data types
- File I/O
- **C++ classes and objects (only for C++)**
- Pointers
- Libraries

Classes

- C++ classes are very similar to Java classes, but still different.
- In C++, code for a class is usually split in two files.
    - Header file (`.h`) contains class fields and member function specifications.
    - Source file (`.cpp`) has function bodies (implementations).
- Other major differences are in OOP functionality.
    - C++ allows multiple inheritance.
    - C++ does not have interfaces. Instead there's something called "abstract class", used in inheritance and polymorphism.
    - C++ allows operator overload `(+, =, >, < , <<,` etc)
    - C++ has destructors. Java has finalize() method but it is executed entirely at the discretion of the Garbage Collector.

Classes

- Construction:
  - C++ allows **overloaded constructors**. Each constructor should have different number and/or type of parameters.
  - Constructor hierarchy (in case of inheritance) is similar to Java though initialization lists do not exist in Java.

- Destruction:
  - Clean up memory and other housekeeping tasks
  - Call when **`delete`** an object for dynamically allocated objects
  - Call when go out of scope of an object for static allocated objects
  - Only one destructor per class, no overload

Objects (e.g., **`IntList`**)

- Similar to Java, an object of a class is created using the **`new`** keyword
  - Static allocation: **`IntList list(3);`**
  - Dynamic allocation: **`IntList* list = new IntList(3);`**
- Accessing data and functions of an object is via the "**`.`**" or "**`->`**" operation
  - Static allocation: **`list.append(10);`**
  - Dynamic allocation: **`list->append(10);`**
- Destroy an object invokes the destructor implicitly
  - Static allocation: the object will be destroy as the program exits the scope of the object
  - Dynamic allocation: **`delete list;`**

Outline

- Compiling and executing a C++ program
- Data types
- File I/O
- C++ classes and objects (only for C++)
- **Pointers**
- Libraries

Pointers

- The most beautiful/difficult thing in C/C++.
  - In fact, every array and class is a pointer in Java.
  - In C/C++ you have to explicitly declare a pointer.
  - The use of pointers in C and C++ are the same.
- A pointer is just an address to a memory location. This can be an address of:
  - Another variable
  - Some dynamically allocated memory
  - Some function
  - **`NULL`** (all lower case **`null`** in Java)
- Example: **`pointer_basics.cpp`**, **`array_pointer.c`** and **`modify_reference.c`**

Pointers

- Declaration: using "*" symbol before variable name.
  - `int * ptr = NULL; //creates pointer to integer`

- Allocation: allocate new memory to a pointer using the keyword `new` in C++ (`malloc` C)

  -
    ```
    int *p = new int;    // pointer to an int  (C++)
    int *p = malloc(sizeof(int));  // (C)
    ```

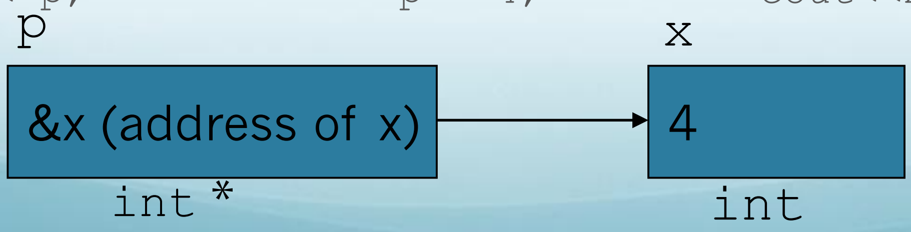
  -
    ```
    int *p = new int[10];    // pointer to an int array (C++)
    int *p = (int *) malloc(10 * sizeof (int));  // (C)
    ```
  - `p` now contains the beginning of the address space for the uninitialized dynamically allocated memory chunk
- Deallocation: clear the allocated memory when you are done using it. Otherwise, **Memory Leak!!!**

  -
    ```
    delete p;   // delete a pointer to a variable  (C++)
    free(p);    // (C)
    ```

  -
    ```
    delete[] p;  // delete a pointer to an array  (C++)
    free(p);    // (C)
    ```

- Dereferencing: accessing data from the pointer

```
int x = 3;          int *p = &x;            cout<<x;
cout<<*p;           *p = 4;         cout<<x;
```



p

&x (address of x)

int *

x

4

int

Pointers - structs and arrays

- Pointers to a struct

```
Student * s = new Student; //a pointer so student struct
(*s).id = 1234; //dereference pointer to access struct fields
s1->id = 1235;              //alternative short-hand way
```

- Pointers to an array

```
int size = 10;     //size of array
int * array = new int[size];    //create array
*array = 1;        // array[0] = 1
array++;            // array[1]
*array = 2;        // array[1] = 2
```

  - The address is incremented by the size of the pointed object.

```
char *p = new char[10];     p++;// advance 1 byte in address
int *p = new int[10];       p++;// advance 4 bytes in address
```

Outline

- Compiling and executing a C++ program
- Data types
- File I/O
- C++ classes and objects (only for C++)
- Pointers
- **Libraries**

C Libraries

- Library reference: http://www.cplusplus.com/reference/clibrary/

Libraries

- C provides a set of standard libraries for

numerical math functions	`<math.h>`
character strings	`<string.h>`
character types	`<ctype.h>`
I/O	`<stdio.h>`

http://www.cs.columbia.edu/~hgs/teaching/ap/slides/CforJavaProgrammers.ppt

Strings

- In C, string is an array of `char` ended with "\0" (a null terminator)
- "hello" = `hello\0`
- Declaring and initialize a string

```
char sstr[10];  // a string of 10 characters
char *str;   // Just a pointer to char
str = "hello";  // now point to a const char*
sstr[0] = str;   // sstr[0] = ‘h’
```

- Copying a string

```
str1 = str2; //shallow copy, both points to the same string
strcpy(s, t);             //deep copy, each has its own copy
sstr = "hello";         // wrong, memory leak!!!  Use strcpy()
```

string.h library

- `#include <string.h>`
- Operations:
  - `char *strcpy(char *dest, char *source)`
    - copies chars from source array into dest array up to NUL
  - `char *strncpy(char *dest, char *source, int num)`
    - copies chars; stops after num chars if no NUL before that; appends NUL
  - `int strlen(const char *source)`
    - returns number of chars, excluding NUL
  - `char *strchr(const char *source, const char ch)`
    - returns pointer to first occurrence of ch in source; NUL if none
  - `char *strstr(const char *source, const char *search)`
    - return pointer to first occurrence of search in source

Formatted strings

- `int sscanf(char *string, char *format, ...)`
  - parse the contents of string according to format
  - return the number of successful conversions
- `int sprintf(char *string, char *format, ...)`
  - produce a string formatted according to format and place this string into the buffer
  - return number of successful conversions

Formatted strings

- Formatting codes for `sscanf`

Code	meaning	variable
%c	matches a single character	char
%d	matches an integer in decimal	int
%f	matches a real number (ddd.dd)	float
%s	matches a string up to white space	char *
%[^*c*]	matches string up to next *c* char	char *

http://www.cs.columbia.edu/~hgs/teaching/ap/slides/CforJavaProgrammers.ppt

Formatted strings

- Formatting codes for `sprintf`
- Values normally right-justified; use negative field width to get left-justified

Code	meaning	variable
%*n*c	char in field of n spaces	char
%*n*d	integer in field of n spaces	int, long
%*n*.*m*f	real number in width n, m decimals	float, double
%*n*.*m*g	real number in width n, m digits of *precision*	float, double
%*n*.*m*s	first m chars from string in width n	char *

http://www.cs.columbia.edu/~hgs/teaching/ap/slides/CforJavaProgrammers.ppt

stdio.h library

- `#include <stdio.h>`
- Formatted I/O
  - `int scanf(const char *format, ...)`
    - read from standard input and store according to format.
  - `int printf(const char *format, ...)`
    - write to standard output according to format
- File I/O: `FILE *`
  - `FILE *fopen(const char *path, const char *mode)`
    - open a file and return the file descriptor
  - `int fclose(FILE *stream)`
    - close the file; return 0 if successful, EOF if not
- Other I/O operations:
  - `int getchar()`
    - read the next character from stdin; returns EOF if none
  - `int fclose(FILE *stream)`
    - close the file; return 0 if successful, EOF if not
  - `char *fgets(char *buf, int size, FILE *in)`
    - read the next line from a file into buf
  - `int fputs(const char *str, FILE *out)`
    - output the string to a file, stopping at '\0'
    - returns number of characters written or EOF

C++ Libraries

- Library reference: http://www.cplusplus.com/reference/clibrary/

STL

- Standard Template Library
  - A set of C++ template classes
    - vector (ArrayList in Java)
    - list (double headed list)
    - stack and queue
    - string
  - Utilities
    - Iterator (iterator and for-each in java)
    - Algorithms: search, count, sort … elements in container classes
- References:
  - http://www.cplusplus.com/reference/stl/
  - http://www.cplusplus.com/reference/algorithm/

Example

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

main()
{
   vector<string> ss;

   ss.push_back("The number is 10");
   ss.push_back("The number is 20");
   ss.push_back("The number is 30");

   cout << "Loop by index:" << endl;

   int i;
   for(i=0; i < ss.size(); i++)
   {
      cout << ss[i] << endl;
    }
...
}
```

http://www.yolinux.com/TUTORIALS/LinuxTutorialC++STL.html#LIST

References

- C++ for Java programmers: http://pages.cs.wisc.edu/~hasti/cs368/CppTutorial/index.html
- C for Java programmers: http://faculty.ksu.edu.sa/jebari_chaker/papers/C_for_Java_Programmers.pdf http://www.cs.columbia.edu/~hgs/teaching/ap/slides/CforJavaProgrammers.ppt
- C for Java Programmers Tutorial http://www.cs.columbia.edu/~hgs/teaching/ap/slides/CforJavaProgrammers.ppt
- C++ Tutorial http://groups.csail.mit.edu/graphics/classes/6.837/F03/lectures/cpp_tutorial.ppt
- A Tour of the Standard Library http://www2.research.att.com/~bs/3rd_tour2.pdf