CMPT 125

Introduction to Computing Science and Programming II

September 8, 2021

General information

Instructor: Igor Shinkar

My email: ishinkar@sfu.ca

Course webpage: *cs.sfu.ca/~ishinkar/teaching.html*
→ Teaching/Seminars → CMPT 125

Office hours:
→ when: Mondays 1PM-2PM
→ where: TASC1 9017

Discussion forum: piazza.com – you should have received an invitation

General information

TAs:

- Sepid Hosseini <sepid_hosseini@sfu.ca>
- Kaumil Trivedi: <kaumil_trivedi@sfu.ca>
- Mohammad Amin Shabani: <mshabani@sfu.ca>

- Email: cmpt-125-help@sfu.ca
- Office hours/format: TBA

General information

Homeworks:

- 5 programming assignments, every ~2 weeks
- 1-2 pen and paper assignments

Midterm: around week 7, during the regular Monday class. TBA

In-lab exam: around week 10, during the regular Tuesday lab. TBA

Final: TBA

General information

https://www.cs.sfu.ca/~ishinkar/teaching/fall21/cmpt125/info.html

- General info
- Lecture notes
- Homework assignments

Lecture notes/exams from last year can be found at
https://www.cs.sfu.ca/~ishinkar/teaching/**fall20**/cmpt125/lectures.html

No need to go to oneclass / coursehero / …

Grading

- Final – 35%
- Midterm – 20%
- In-lab exam – 20%
- Homeworks - 25%

Prerequisites – CMPT 120

- You are expected to be familiar with basic concepts of programming
  - ✓ Variables
  - ✓ Data types (integer, float, char, string)
  - ✓ Lists/arrays
  - ✓ Basic I/O
  - ✓ Conditionals (if/else)
  - ✓ Loops (for, while)
  - ✓ Functions, passing parameters
  - ✓ The [develop->test->debug] cycle

Some history (CMPT 127)

Courses were coupled:

- CMPT125 was more theory
- CMPT127 was labs

Starting this year:

- We have only one course
- The material for the tutorials will be often borrowed from CMPT127
- We‘ll see how things go...
- Your feedback will be important

Learning Outcomes

By the end of these two courses you will learn

- How to write high quality code in C
- Basic algorithms
- Data structures
- Measuring performance of algorithms
- Basic concepts of OOP in C++

Coding

In lectures:

- I will code using replit.com

In the lab:

- You will learn to compile code in Linux using gcc / Makefile
- You will use an IDE that allows debugging with breakpoints, running code step-by-step, inspect variables during the execution, etc...

For homework assigment:

- You will need to submit code that compiles without warnings in CSIL
- The assignments will contain the exact instructions.

Questions so far?

Computer science review

Algorithms are the core component

An algorithm is a sequence of instructions (recipe) for solving a problem, i.e., obtaining a required output for a valid input.

- We communicate algorithms to computers using programming languages.
- In this course the programming language will be C

(and some C++).

Today

Crash course in C

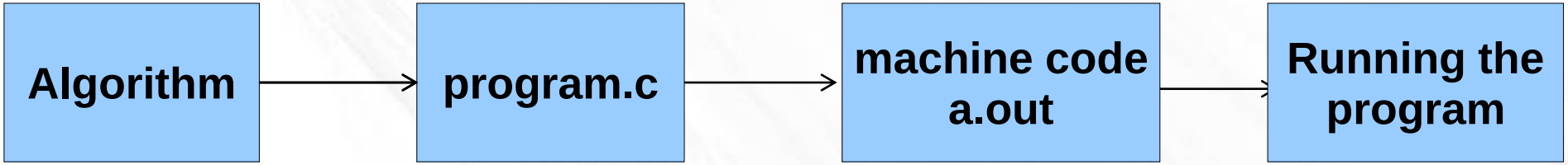
- Differences between C and Python
- Compilation process
- Variables, strong typing
- Memory model: data vs address

Compilation process

In Python:

Interpreter runs one instruction at a time

In C/C++



Code is compiled into machine language in advance

Compilation process

```python
#!/usr/bin/env python
import sys
import os
import simpleknn
from bigfile import BigFile

if __name__ == "__main__":
    trainCollection = 'toydata'
    nimages = 2
    feature = 'f1'
    dim = 3

    testCollection = trainCollection
    testset = testCollection

    featureDir = os.path.join(rootpath, trainCollect
```

In Python:

- Interpreter runs each instruction at a time
- Errors are caught only when the interpreter is trying to run them

In C/C++:

- Code is compiled into machine language in advance
- Some errors are caught at compile time
- Faster performance (e.g. due to memory allocation)

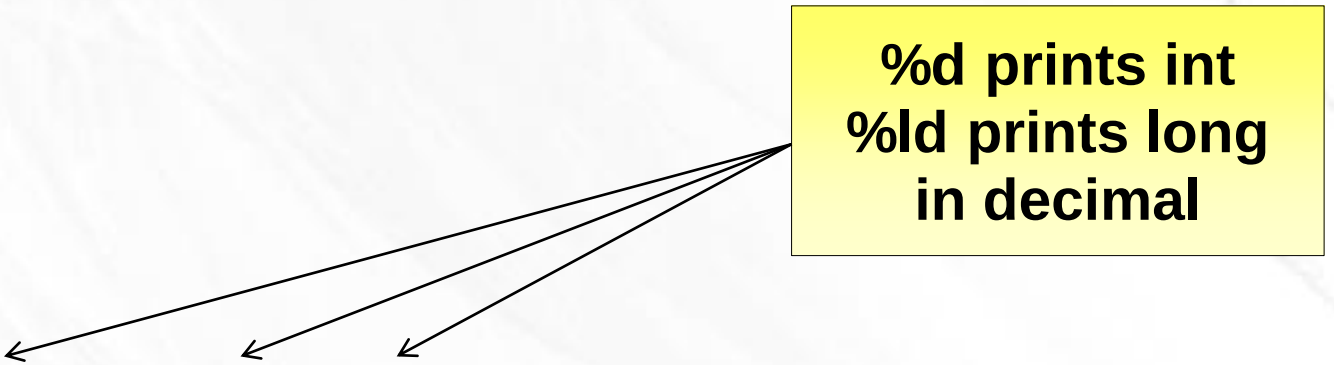
```c
#include <stdio.h>

#define SIZE 4

int main()
{
    char *strings[SIZE] =
                    {
                      "String1",
                      "String2",
                      "String3",
                      "String4"
                    };

    char *ptr_swap;   /* A temporary pointer to swap strings */

    /* Swap "String2" with 'String3' */
    ptr_swap  = strings [1];
    strings [1] = strings [2];
    strings [2] = ptr_swap;
    printf("%s %s %s %s", strings[0], strings[1], strings[2], strings[3]);

    return 0;
}
```

Variables

Declaring Variables

- In C all variables must be declared before using them.
- The type of variable must be declared.

```
int main()
{
   int x = 5;
   long y = 10;

   printf("The sum of %d and %d is %d", x, y, x+y);

    return 0;
}
```



>> The sum of 5 and 10 is 15

Strong Typing

- In C the type of variable cannnot be changed.
- Once it is declared as int, it will always stay int.
- (It is possible to change types in Python)

- When a variable is declared, a space in memory for the variable is reserved. For example:
  - int - 4 bytes (sometimes 2 bytes, compiler dependent)
  - long – 8 bytes
  - double – 8 bytes
  - char – 1 byte

Pointers and References

Pointers and References

- Each variable is stored in a unique location in the memory.
- Its address is represented by a number (can be accessed using pointers and references).
- Thus, a variable has 3 main features:
  - type
  - value
  - address in memory
- Address can be store in a variable explicitly

```
int x = 5;
int* px = &x;  // pointer to the location of x
printf("The address of %d is %p", x, px);
>>    The address of 5 is 0x9a58af3c4
```

%p prints the address (value of the pointer)

Pointers and References

Dereferencing:

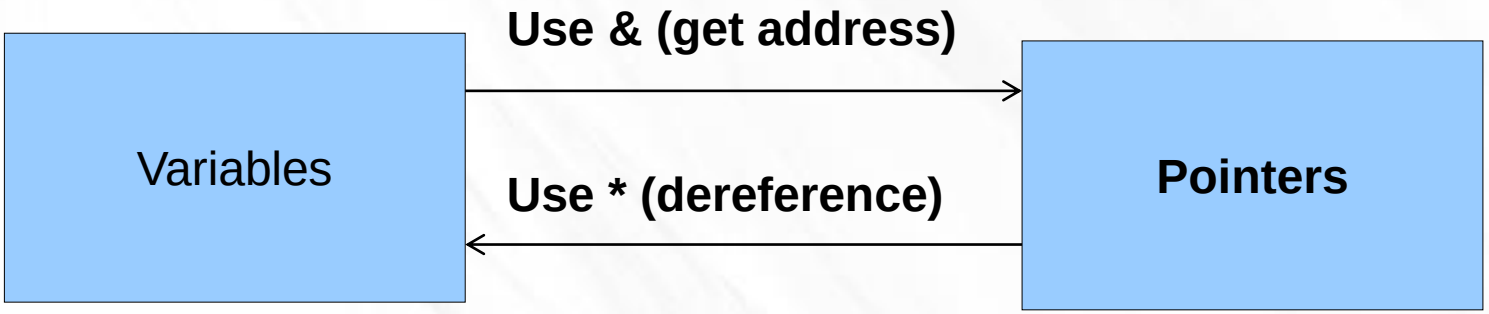
```
int x = 5;   // variable x has value 5
printf("x = %d", x);

int* px = &x; // pointer to the location of x
printf("The value at the address %p is %d", px, *px);


*px = 7;  // dereferencing, changing the value at the address px
printf("x = %d", x);
```

```
>>      x = 5
>>      The value at the address 0x73b8df7a3 is 5
>>      x = 7
```

Pointers and References

Remember the difference:

- Variable – the data
- Pointers – the address



Why are pointers useful?

Allows us to modify parameters in a function

```
void swap(int a, int b) {
        int tmp = a;
        a = b;
        b = tmp;
}

int main() {
        int a = 2;
        int b = 3;
        swap(a,b); // the values will not change!!
        ...
```

Why are pointers useful?

Allows us to modify parameters in a function

```
void swap(int* a, int* b) {
        int tmp = *a;
        *a = *b; // dereferencing
        *b = tmp; // dereferencing again
}

int main() {
        int a = 2;
        int b = 3;
        swap(a,b); // WRONG!!! wrong type of parameters
                                (though it will compile on some compilers)
```

Why are pointers useful?

Allows us to modify parameters in a function

```
void swap(int* a, int* b) {
        int tmp = *a;
        *a = *b; // dereferencing
        *b = tmp; // dereferencing again
}

int main() {
        int a = 2;
        int b = 3;
        swap(&a, &b); // the values will change!!

}
```

Why are pointers useful?

- Allows modifying parameters in a function
- Allows us to pass large objects to a function with a single pointer
- Optimization - avoids duplicating data
- Arrays
- Strings

Arrays

Arrays

- An array represents a list of elements
- The list is of a fixed length – once created cannot be resized
- All elements have the same type
- Access by array[index]
- The indexing is [0]..[length-1]

Arrays - example

```
int main() {
        int array[7];
        for (int i=0; i < 7; i++) {
                array[i] = i+5;
        }
        printf("array[3] = %d\n", array[3]); // array[3] = 8
        array[3] = 66;
        printf("array[3] = %d\n", array[3]); // array[3] = 66
        …
}
```

Initializing arrays at declaration

```
int main() {
        int array[7];
        for (int i=0; i < 7; i++) {
                array[i] = i+5;
        }

        …

}
```

OR

```
int main() {
        int array[7] = {5, 6, 7, 8, 9, 10, 11};
```

Iterating through an array

```
int main() {
        int i;
        int array[10] = {0, 1, 8, 2, 18, 3, 6, 2, 2, -4};
        for (i = 0; i < 10; i++)
                printf( "array[%d] = %d\n", i, array[i] );

        for (i = 0; i < 10; i++)
                printf( "array[%d] = %d\n", i, *(array+i) );
}
```

Array – represenation in C

The variable of type *array of ints* is really a *pointer to int*.

The elements are stored in the memory in a contiguous block starting from the position array.

Arithmetics of pointers: Note that size of int = 4

This means that *array+i* really increases by *i*sizeof(int)*.

```
int array[10] = {6, 1, 8, 2, 18, 3, 2, 2, -4};

int* array_ptr = array;
```

Both point to element 6 in the array

Iterating through an array

```
int main() {
        int i;
        int array[10] = {0, 1, 8, 2, 18, 3, 6, 2, 2, -4};
        for (i = 0; i < 10; i++) {
                printf("array[%d] = %d\n", i, *(array+i));
        }
}
```

Iterating through an array

```
int main() {
        int array[10] = {0, 1, 8, 2, 18, 3, 6, 2, 2, -4};
        int* first = array;
        int* last = array + 9;
        int* iter;
        for (iter = first; iter <= last; iter++)
                printf("%d is at the address %p \n",  *iter, iter);
        ...
```

Changing values in an array

```
int main() {
        int array[10] = {0, 1, 8, 2, 18, 3, 6, 2, 2, -4};
        printf("array[3] = %d\n", array[3]); // array[3] = 2;
        array[3] = 66;
        printf("array[3] = %d\n", array[3]); // array[3] = 66;
        *(array+3) = 25;
        printf("array[3] = %d\n", array[3]); // array[3] = 25;
```

Array bounds

```
int array[10] = {0, 1, 8, 2, 18, 3, 6, 2, 2, -4};
```

Q: What happens when trying to access array[-1] or array[10]?

A: Will return *garbage data* or *crash*

Using pointers to access an array

```
int main()  {
        int arr[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
        int* ptr;

        ptr = arr+3;
        printf("*ptr  = %d\n",  *ptr);

        ptr = ptr+2;
        printf("*ptr  = %d\n",  *ptr);

        ptr = &arr[9];
        printf("*ptr  = %d\n",  *ptr);

        arr = &arr[5];
    }
```

NO!! Array cannot be reassigned.
Array is a *constant* pointer

Arrays - recap

- An array represents a list of elements
- The list is of a fixed length
- All elements have the same type
- Access by array[index]
- The indexing is [0]..[length-1]
- The variable of type *array of ints* is really a *constant pointer to int*.
- Can use pointers to access an array

Questions?

Comments?