Intro. to Computer Science/OOP [CSC-120] - Review Guide

GitHub Repository

Process of Writing Code

- Source Code: The code you're writing
- Pre-Processor: Including "codes" in your source code (i.e. `#include <iostream>`)
- Process: Source Code → Pre-Processor → Compiler → Binary Code → Linker → Executable File

Elements in the Program

- Keywords/Reserved Words (i.e. int, double, float, char, string, bool) → Comes with the language
- Programmer-Defined Identifiers (i.e. Variable Names)
- Operators (i.e. `<<`)
- Punctuations (i.e. comma, semi-colon)
- Syntax
- Main Function (Entry Point/Driver Function)

Errors

- Pre-Processor Error: Maybe file doesn't exist?
- Syntax Error: Compile Error
- Run-Time Error: Only happening when the code is running (i.e. dividing by 0)
- Logical Error: Code isn't running as expected

Input/Output

Cin & Cout

- Stands for **"Character Input/Output"**

```
#include <iostream>
using namespace std;

```

```
int main() {
    int num1, num2;
    cin >> num1 >> num2;
    cout << num1 + num2 << endl;
}
```

Escape Character

IS A CHARACTER

EX:

- \n: New Line
- \t: Tab
- \b: Blank

```
#include <iostream>
using namespace std;

int main() {
    cout << "Hello World!\n";
    cout << "Hello World!" << endl;
}
```

Both will do the same thing, which is switching to a new line.

Reading & Writing Files

Use `#include <fstream>` or file access

- Input: ifstream
- Output: ofstream

```
#include <iostream>
using namespace std;

int main() {

    //Declaring variables.
    ifstream inFile;
    ofstream outFile;
}
```

Opening Files:

```
#include <iostream>
```

```
using namespace std;

int main() {

    ifstream inFile;
    inFile.open("text.txt"); //File name can include the whole path.

    //See if the file exists.
    if (!inFile) {
        cout << "File doesn't exist!" << endl;
    }
}
```

Data Types

Integer (int)

- Between **about** -2B ~ 2B
- 4 Bytes (32 bits)
- By using **unsigned** → all positive number → double the "space"

Float (float)

- 4 Bytes (32 bits)
- Holds decimal places (not usually used)

Double (double)

- 8 Bytes (64 bits)
- Also holds decimal places like float, more commonly used

Character (char)

- 1 Byte (8 bits)
- Total Character Number: 2^7 = 128
- ASCII Code: Each character has a matching number

Boolean (bool)

- 1 Byte (8 bits)
- Can only be **true or false**
- Things other than 0 will be false; otherwise, true

Operations

Modulus (%)

- Find the remainder of the division
- Cannot do with decimals

Division

```
#include <iostream>
using namespace std;

int main () {
    cout << 13/5 << endl; //Prints 2

    cout << 13/5.0 << endl; //Prints 2.6
}
```

Assignments & Initialization

Initialization

First time giving a value.

```
#include <iostream>
using namespace std;

int main() {
    int x = 10;
}
```

Assignment

Second+ time giving a value again.

```
#include <iostream>
using namespace std;

int main() {
    int x = 10; //Initialization

    x = 20; //Assignment
}
```

Scope

- Main is a scope, where variables declared inside cannot be accessed outside the scope
- Scope of a variable: the part of the program in which the variable can be accessed

```cpp
#include <iostream>
using namespace std;

int main() {
    int x; // Can only be recognized in main().
}
```

Condition

If-Else Statement

Asking questions

EX: If a variable is equaled to a certain value

The result of a condition is always true or false

```cpp
#include <iostream>
using namespace std;

int main() {
    int income = 1000;

    //Is x equaled to 100?

    if (income > 1000) {
        cout << "You are rich." << endl;
    }
    else {
        cout << "You are poor." << endl;
    }

    //Going to print "You are poor." because income isn't greater than 1000.
}
```

Only If

```cpp
#include <iostream>
using namespace std;

int main() {
    int income = 1000;

    if (income > 1000) {
        cout << "You are rich." << endl;
    }

    //Nothing will be printed because the if is passed.
}
```

Nested-If

If inside of an if statement (or more)

```cpp
#include <iostream>
using namespace std;

int main() {

    int score = 100;

    if (score >= 94) {
        cout << "You get an A!\n";

        if (score == 100) {
            cout << "You also get a full!\n";
        }
    }
}
```

Switch Statement

```cpp
#include <iostream>
using namespace std;

int main() {

    char grade = 'A';

    switch (grade) {
        case 'A':
            cout << "Congrats! You get an A!\n";
            break;
        case 'B':
            cout << "Nice! You get a B!\n";
            break;
        case 'C':
            cout << "Try harder! You get a C!\n";
            break;
        case 'D':
            cout << "Oh no! You get a D!\n";
            break;
        case 'F':
            cout << "Unfortunately, you failed.\n";
            break;
        default:
            cout << "Invalid grade.\n";
    }
}
```

Relational Operators

Operators	Meaning
>	Greater Than
<	Less Than
>=	Greater Than & Equal To
⇐	Less Than & Equal To
==	Equal To
≠	Not Equal To

Logical Operators

Operator	English	Meaning
&&	AND	New Relational Expressions will be true if both conditions are true. Otherwise, it's false.
||	OR	New Relational Expressions will be true if either of the conditions is true. If both are false, it's false.
!	NOT	Opposite of the expression.

Relational Expressions

Result of condition can be assigned as variable (boolean)

```
#include <iostream>
using namespace std;

int main() {
    bool result;

    int x = 10, y = 20;

    result = x == y;

    /*
    x == y is a conditional statement, which will return either true
    or false.
    */

    //In this case, result will be false since x != y.
}
```

Conditional Operators

Shortens the if-else statements

Format: expression ? expression : expression;

```
#include <iostream>
using namespace std;

int main() {

    int x = 10, y, z;

    //If x > 0, y = 10. Else, z = 10.
    x > 0 ? y = 10 : z = 10;

    //Same code.

    if (x > 0) {
        y = 10;
    }
    else z = 10;

    return 0;
}
```

Flags

Usually implemented as "bool"

- 0 = false
- non-0 = true

Loops

While Loop

- Executes "same code" for many times
- Not sure how many times the loop will run

- Statement will return true or false

For Loop

Knows **exactly** how many times to execute code

3 Expressions in For Loop

- Initialization –> Start with a number (counter)
- How many times? (Condition)
- Plus 1 (or other statements)

```
#include <iostream>
using namespace std;

int main() {
    //Prints "My name is Sean!" 10 times.
    for (int i = 0; i<10; i++) {
        cout << "My name is Sean!" << endl;
    }

    //Initialization: int i = 0;
    //How many times? (Condition): i<10;
    //Plus 1 (or other statements): i++
}
```

Do-While

Basically the same as while but **guarantee** at least 1 execution

Nested Loops

A nested loop is a loop inside the body of another loop

Inner (inside), outer (outside) loops:

```
#include <iostream>
using namespace std;

int main() {

    //This loop will be executed 9 times.
    for (int i = 0; i<3; i++) {
        for (int j = 0; j<3; j++) {
            cout << "HI\n";
        }
    }
}
```

Range Loop

```
#include <iostream>
using namespace std;

int main() {
    string text = "text";

    cout << "Characters in the string are: \n";
    for (char ch : text) {
        cout << ch << '\n';
    }
}
```

Functions

EX:

```
void add() {
    int a = 1, b = 2;
    cout << a + b << endl;
}
```

- Functions that don't return a value are declared as "void"
- Functions that return values are declared whatever value they return
- Function parameters are considered local

Parts of Functions

1. Prototype (declaration of the function)
2. Definition: What does the function do?
3. Call of function

Array

List/Collection of data

Reserving the numbers of bytes needed for the array in RAM **continuously**

Size of the array should **always** be declared as a constant variable before declaration of the array

```
#include <iostream>
```

```cpp
using namespace std;

int main() {

    //5 grades --> 20 bytes reserved in memory
    int grades[5];

    //Index starts from 0 --> 4 Index
    grades[0] = 100;
    grades[1] = 90;
    grades[2] = 80;
    grades[3] = 70;
    grades[4] = 60;

    //Accessing elements from array:

    cout << grades[0] << endl; //Printing out the first element in the grades array.

    return 0;
}
```

- Global Array → All elements initialized to 0
- Local Array → Uninitialized
- Static Array → Doesn't change size (in C++).

Sizeof

Know the size of the array

```cpp
#include <iostream>
using namespace std;

int main() {
    const int SIZE = 5;
    int nums[SIZE];
    cout << "Size of the array: " << sizeof(nums) / sizeof(int) << '\n';
    return 0;
}
```

In functions, since we're only passing reference that's pointing to the first element of the array, sizeof won't work in functions. Therefore, we'll need to include a size variable for the function.

Vector

- Dynamic array → Not static

- `#include <vector>` → Doesn't come with C++, like string
- Using an array behind the scene

```
#include <iostream>
#include <vector> //Include the file
using namespace std;

int main() {
    //Put whatever that's storing inside the vector inside < >.
    vector<int> nums;

    //To "add" elements into vectors, use "push_back()".
    nums.push_back(1);

    return 0;
}
```

Behind the Scene - Vector

- Initially creates an array of 10 elements
- Has a variable called capacity → starts with 10
- Using the capacity variable, creates an array
- Has a size variable → Keep track of how many elements inside vector → Starts with 0
- Increases size by 1 with each push_back
- Once capacity is full → Doubles capacity → Creates new array with new capacity → Copies elements from old array to new array

Pointer

Address of a variable (location in memory)

```
#include <iostream>
using namespace std;

int main() {
    int x = 20;

    //Using &, print the address of x in memory.
    cout << &x << endl;
}
```

Pointer Variables

Store the address of a variable into a variable (pointer)

```
#include <iostream>
```

```cpp
using namespace std;

int main() {
    //To declare a pointer, put a *.
    int *x = 20;

    //20 will be stored somewhere in the memory.
    //x will be holding where 20 is in the memory.

    //Printing x --> The address that the pointer is holding.
    cout << x << endl;

    //Printing the address of x.
    cout << &x << endl;

    //Printing the value that x is pointing to, which is 20.
    cout << *x << endl;
}
```

Size of all pointers are **8 Bytes**.

Can initialize pointers into **nullptr**.

Search Algorithm

Find if an element exists in an array.

```cpp
#include <iostream>
using namespace std;

//Time Complexity: O(n)
bool find(int arr[], const int SIZE, const int TARGET) {
    for (int i = 0; i<SIZE; i++) {
        if (arr[i] == TARGET) {
            return true;
        }
    }
    return false;
}

int main() {
    const int SIZE = 10;

    //Declare an array.
    int arr[SIZE] = {2, 4, 6, 8, 9, 13, 15, 16, 20, 22};

    //See if 20 is inside of the array.
    bool has20 = find(arr, SIZE, 20);
}
```

Time Complexity

Time of algorithm (sort of)

Search Algorithm: Linear –> go from start to end –> O(n)