**CS121 SI Week 1 Worksheet – Pt. I Solutions**

Concept & Syntax Questions:

1. What are the three types of loops used in C++? How does each differ from one another? What are some example scenarios of when each is preferred?

|  |  |  |
| --- | --- | --- |
| **Loop Name** | **How it works** | **Example Scenarios** |
| While loop | Runs 0 or more times | Games, File IO |
| Do... while | Runs at least once | Menus, Input Validation |
| For | Runs x amount of times | Summation, Fizzbuzz |

**BONUS:** Can each loop replace the others? Explain. Yes they can replace each other.

|  |  |  |  |
| --- | --- | --- | --- |
| **Loop Name** | **Replace While** | **Replace Do... while** | **Replace For** |
| While loop | -- | Make condition always true on first run of loop | Declare a counter and add update inside loop |
| Do... while | Add an if-statement | -- | Add if-statement and do same as above |
| For | Omit init and update:  *for(; conditional ;)* | Same as while loop but make sure condition is always true at start | -- |

<Examples of the above bonus answer are in a file on the GitHub repository>

1. What's the difference between an if/else chain and a switch statement? When would either be preferred?

If/else chains can work with any conditional statement. Use them with ranged-based work (e.g. if(grade > 90 || grade <= 100) ) and usually anything that isn't a menu.

Switch statements (in C++) only work with integer-based variables and can only check by equality (cases). They work best in menus (e.g. choice == OPEN, choice == CLOSE, etc.).

**BONUS:** Can an if/else chain replace a switch statement? Can the reverse occur? Explain.

Yes, if/else can replace switch statements. They can handle case-by-case equality checks.

No, switch statements cannot replace if/else. They can only handle integer-based variables.

1. If you were asked to make a program that managed a football team's information (e.g. individual player statistics, team roster, team name, and so on), what programming concepts could be applied?

**HINT:** One answer would be something like "if statements".

More examples: Functions (perform tasks), loops (for menus), and arrays (list of members).

1. Write a code segment to generate a (psuedo-)random integer between 1 and 100, another between 4 and 20, and a last one between 1900 and 2014.

// NOTE: rand() % range + start\_val; is the general equation to generate a random integer

// range is defined as: end\_val – start\_val + 1 (e.g. 1 through 5 is 5 – 1 + 1 = 5)

// start\_val is the starting value of your range

srand( time(NULL) ); // initialize seed value (or else same rand num each prog. run)

int rand1 = rand() % 100 + 1, // 1 through 100

rand2 = rand() % 17 + 4, // 4 through 20

rand3 = rand() % 115 + 1900; // 1900 through 2014

**BONUS**: What library is required to use the "rand()" function?

*#include <cstdlib> //c-standard library*

1. What are two reasons we should use both descriptive, yet simple/short identifiers?

Reduce complexity and improve readability.

1. What is the output of the following statements when ran?

**HINT:** Think about operator precedence (i.e. rules for which operator acts before/after others).

bool a, b, c, d;

a = c = true;

b = d = false;

cout << (a == c) << endl; true or 1

cout << (!a == d) << endl; true or 1

cout << (0 == a) << endl; false or 0

cout << (!a && !b || !c && !d) << endl; false or 0

cout << ((true == b) || -1) << endl; true or 1

cout << (!a || b && a && c || !d) << endl; true or 1

**BONUS:** Could you avoid including "using namespace std;" in the above code? Explain.

No, unless you replace all *cout* and *endl* with *std::cout* and *std::endl*. Each object is defined in the standard namespace, and without identifying that the compiler doesn't know what it is.

Practice Project(s): Below is posted on the GitHub page.

1. Write a code segment (in C++) that asks a user for a temperature and if it is in Fahrenheit or Celsius. If it is in Fahrenheit, print out its Celsius equivalent (and otherwise if it is in Celsius initially).

**BONUS**: Make functions for the integer grab and each conversion. You should have three to four functions in total (depends on if you include main).

1. Write a code segment that asks the user for an integer to compute the factorial of a number and print out its factorial. If the integer is negative, print an error message.