**C++ Reference - Past Problems**

1. **How to run**

**Mac**

Xcode: Make a new project and put the file in.

In Terminal:

1. cd to the dir where the .cpp

2. g++ main.cpp -o main.out (compile with gcc or g++ and output on the main.out)

./main.out (run the output)

**Windows:**

Visual Studio : Make a new project and run the file there, like xcode.

**++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++**

1. **Samples**

**//------------------------------------------------------------------------------------------------**

// character input to function - Implementation 1

float print\_char(char \*msg) //Or msg[ ]

{

cout << msg;

...

}

main(){

char message[] = "Your Message";

print\_char(message) //no \* or [ ]. Just the name.

...

}

**//------------------------------------------------------------------------------------------------**

The Deprecated conversion from string literal to 'char\*' WARNING:

The strings that you enter: "red", "organge" etc are "literal", because they are defined inside the program code itself (they are not read directly from disk, user input /stdin etc.).

This means that if at any point you try to write to your colors you will be directly accessing your original input and thus editing it. This would cause some undesired run-time errors.

Declaring it as a const will make sure that you will never try to write to this pointer and such a run-time error can be avoided.

const char \*colors[4] = {"red", "orange", "yellow", "blue"};

If you ever feel like editing these values at runtime, then you should copy the strings first.

**//------------------------------------------------------------------------------------------------**

//Pointers and char strings

//A type char variable is actually a pointer to the first letter. Thus, we can store a string in a pointer

int \*s,

s = "foobar";

cout<<\*s; // Will print: f

cout<<\*s++; //Will print: o

//The last character of every string is '/0'. That in case you wanna print the full string letter by letter.

**//------------------------------------------------------------------------------------------------**

**Pointer memory should be allocated prior to use.**

void main() {

int\* x; // Allocate the pointers x and y

int\* y; // (but not the pointees).

x = new int; // Allocate an int pointee and set x to point to it.

\*x = 42; // Dereference x and store 42 in its pointee

\*y = 13; // CRASH -- y does not have a pointee yet

y = x; // Pointer assignment sets y to point to x's pointee

\*y = 13; // Dereference y to store 13 in its (shared) pointee

}

**//------------------------------------------------------------------------------------------------**

**Function parameter by reference (only in cpp)**

// Passing parameters by reference. **Here, the symbol & practically does not mean "address". It means that f.i. is an alias of the variable passed as an argument. Also, see, that dereferencing is not needed within the function.**

#include <iostream>

using namespace std;

void Duplicate(int& a, int& b, int& c) {

a \*= 2;

b \*= 2;

c \*= 2;

}

int main() {

int x = 1, y = 3, z = 7;

Duplicate(x, y, z);

// The following outputs: x=2, y=6, z=14.

cout << c"x="<< x << ", y="<< y << ", z="<< z;

return 0;

}

**//------------------------------------------------------------------------------------------------**

**Function parameter by reference 2 - the pure C way**

void Duplicate(int \*a, int \*b, int \*c) {

\*a \*= 2;

\*b \*= 2;

\*c \*= 2;

}

int main() {

int x = 1, y = 3, z = 7;

Duplicate(&x, &y, &z);

// The following outputs: x=2, y=6, z=14.

cout << "x=" << x << ", y=" << y << ", z=" << z;

return 0;

}

**//------------------------------------------------------------------------------------------------**

**//------------------------------------------------------------------------------------------------**

**Questions to Antonis:**

* Notice with C++ references, we do not need to pass the address of a variable, nor do we need to dereference the variable inside the called function (???)