OOP344 Test 1 V1

First Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Last Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Section: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Student ID: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Please answer each question as concisely as possible in the test booklet provided. This test is worth 15% of your final mark. Please ***print*** your answers as neatly as possible, you will not be given a mark if I cannot read your answers. You may have 2 reference materials, which can be any combination of class textbook, or class notes (bound together in a binder).

Section 1 - Coding (25 Marks)

1) Coding a Stack [6 Marks]  
Consider the following code:

struct Node {

  Product\* data;

  Node\* next;

};

class Stack {

  Node\* front;

  public:

  void push(Product\*);

  Product\* pop();

};

Code the push and pop functions for the Stack class so that this class would work as a stack would be expected to work. The push function will add a new value to the stack, and the pop function would remove a value from a stack, returning that value.

2)  Writing a product base class [10 Marks]

(global variables will not receive a mark)

Write a class that will encapsulate the functionality of a product for an inventory system. This class will be called Product. The class requires the following private member variables:

* An unique product ID
* A product Name

The class also requires the following public functions:

* A constructor that can receive a null terminated character string or no arguments at all. The constructor allocates a character array to store the incoming string and copies the values of the string into the new array, setting this as the product’s name. If no string is provided, product name is set to null. The constructor also sets the product ID to a new numeric ID that cannot be the same as any other product instance’s ID. It is expected that this value will increment with each product instantiated (the first product instance’s ID is 0, the next is 1, the next is 2, etc)
* A virtual destructor that deallocates the product’s name if it is not null
* A constant function that returns a constant character pointer to the product’s name
* A pure virtual function called setInfo that receives 1 float and 1 to 5 integers, if the integers are not provided they should default to 0
* An overloaded dereference operator (unary \*) that returns a reference to the product instance. When this operator is used, a message is written to the console that is formatted as: “[product name] with an ID of [product ID]” (items in [ ] square braces should be replaced with their correlating value)

3) Inventory Program [9 Marks]

Consider the following definitions (assume these are fully implemented classes):

class Furniture : public Product {...};

class Electronic: public Product {...};

class Grocery: public Product {...};

Write a inventory program that stores a collection of products in a stack (use the stack written above). First, the program should ask what the user wants to do. The program should then do one of the following depending on the input:

“Exit” - Exits the program

“Deliver” - Pops the next product off the list, prints to the screen “[product name] with

an ID of [product ID] is ready to be delivered” (you’ll need to use the dereference operator on the product to print the first part), then deletes the product

“Receive” - See below

When the user enters “Receive”, the program should then prompt the user for what type of product they will receive, the options are “Furniture”, “Electronic” or “Grocery”. The program will then prompt the user for information about the object which should follow the format of:

“[product name] [float] [int] [optional int] [optional int] [optional int] [optional int]”

So for example:

“Gameboy 2.3 10 2 4 3”

The program will then allocated a new object that correlates with the type entered earlier, providing it the name above. The numbers after the name should be split up and sent to the product’s setInfo function (example above could be “setInfo(2.3, 10, 2, 4, 3)” or “setInfo(2.3, 10, 2, 4, 3, 0)”). The new product is then pushed onto the stack.

Finally, the program should loop back and ask the user for what they want to do again unless the program is supposed to exit.

Section 2 - Walkthrough (10 Marks)

Walkthrough the following code and write out the EXACT output of the program with the following command line arguments:  
  
$a.out one two three

#include <string.h>

#include <iostream>

#define DOWN

namespace Quick

  {

  typedef float retVal;

  void out(retVal a) {std::cout << a << std::endl;}

  retVal fire(char\* a)

     {

     if(strcmp(a, a+1) > 0)

        return (retVal)10 / 4;

     else

        return (retVal)4 / 10;

     }

#define OUT(a) {std::cout << a << std::endl;}

  }

namespace Slow

  {

  typedef int retVal;

  void out(retVal a) {std::cout << "The output is: \"" << a << "\"" << std::endl;}

  retVal water(char\* a)

     {

     if(strcmp(a, a+1) > 0)

        return (retVal)10 / 4;

     else

        return (retVal)4 / 10;

     }

#define OUT(a) {std::cout << "The output is: \"" << a << "\"" << std::endl;}

  }

int lightning(char\* a)

  {

  char\* b = a;

  while(\*b) b++;

  return b-a;

  }

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

  {

  void\* funcs[] = {Quick::fire, Slow::water, lightning};

#ifndef DOWN

  for(int i = 0; i < argc; i++)

#else

  for(int i = 3; i >= 0; i--)

#endif

     {

     if(!strcmp(argv[i], "one"))

        {

        auto r = (\*(Slow::retVal(\*)(char\*))(funcs[1]))(argv[i]);

        Slow::out(r);

        }

     else if(!strcmp(argv[i], "two"))

        {

        auto r = (\*(Quick::retVal(\*)(char\*))(funcs[0]))(argv[i]);

        Quick::out(r);

        }

     else if(!strcmp(argv[i], "three"))

        {

        auto r = (\*(int(\*)(char\*))(funcs[2]))(argv[i]);

        OUT(r);

        }

     else

        {

        OUT(argv[i]);

        }

     }

  return 0;

  }