**Prokon C++ Competency Test**

**Question 1**

Consider the following code segment, state which statements are incorrect and motivate your answers.



\*j = 10

j has not yet been assigned an address, so it cannot be assigned a value.

\*k = &j

One cannot assign a two dimensional pointer to a one dimensional pointer. I.E. \*\*int cannot be assigned to \*int

J \*&\*\*m;

There is no operand between J and M. The values are incompatible, as J is int\* and m is int\*\*

Delete &i

One cannot delete an address, as it does not have a destructor

m[z] = new int(9);

Cannot assign a value of int\* to int\*\*

**Question 2**

Assume the following code segment and answer the questions that follow:



1. Write code that allocate *array3D* such that it will become a dynamic 3-dimensional array of **int**’s, where *length (1st dimension) = width (2nd dimension) = height (3rd dimension) = n.* Each **int** should be initialized to 7.
2. Write code to allocate *array2D* such that it will become a dynamic 2-dimensional array of **int pointers**, where *length (1st dimension) = width (2nd dimension) = n*. Each cell in the array should be initialized to a dynamically allocated **int** value of 5.
3. Write code to de-allocate *array2D.*

1

for (int i = 0; i < n; ++i)

{

array3D[i] = new int\*[n];

for (int j = 0; j < n; ++j)

array3D[i][j] = new int[n];

}

for (int i = 0; i < n; ++i)

{

for (int j = 0; j < n; ++j)

{

for (int k = 0; k < n; ++k)

{

array3D[i][j][k] = 7;

}

}

}

2

int val = 5;

array2D = new int\*\*[n];

for (int i = 0; i < n; ++i) {

array2D[i] = new int\*[n];

}

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

array2D[i][j] = &val;

}

}

3

for (int i = 0; i < n; i++)

{

delete[] array2D[i];

}

delete[] array2D;

**Question 3**

Assume the following code segment and answer the questions that follow:



1. Implement the = operator for the *Basket* class.

void operator=(const Basket &right)

{

delete[] fruit;

totalWeight = right.totalWeight;

numItems = right.numItems;

size = right.size;

}

1. Provide the declaration (signature) of the += operator as it would appear in the *Basket.h* file to add new fruit to the basket as follows:



Basket& operator+=(const Basket & right)

1. Implement the += operator declared above. A fruit should be added only if there is space in the basket. If there is no space, do nothing The += operator must keep track of the total weight and the total number of fruit.

Fruit& operator+=(const Fruit & right)

{

int availableSpace = numItems - maxSize;

int insertIndex;

Fruit currentFruit;

if (availableSpace > 0)

{

insertIndex = availableSpace + 1;

for (int i = 0; i < availableSpace; i++)

{

currentFruit = right.fruit[i];

fruit[insertIndex] = currentFruit;

numItems += 1;

totalWeight += currentFruit.weight;

insertIndex += 1;

}

}

}

**Question 4**

Consider the following class definitions and answer the questions that follow:



1. Which of the classes above are abstract?

Monster, FlyingMonster

1. Write down the code for *Dragon* class’ constructor.

Dragon(int name, int maxFire = 5, int maxFlying = 5)

{

this->name = name;

maxFireDamage = maxFire;

this->maxFlying = maxFlying;

}

1. Assume a *Dragon* object is initialized as follows:

If the function call is valid, which class’ version of the function will be used in each of the function calls: *Monster, FlyingMonster, Dragon or HorseFly*?

1. dragon.fly(); Dragon
2. flyingMonster->fly(); *FlyingMonster*
3. cout<<dragon.getName() <<endl; Monster
4. flyingMonster->attack(); Dragon
5. flyingMonster->breatheFire(); Dragon
6. For each of the following statements, provide a reason why the statement is invalid:
   1. Dragon \*smaug = new FlyingMonster("Smaug", 70);

Cannot instantiate instance of virtual class

* 1. HorseFly thatAnnoyingFly(1000000, "TabanusAtratus");

Monster \*monster = thatAnnoyingFly;

Horsefly inherited protected from FlyingMonster, meaning its base classes cannot be accessed

* 1. FlyingMonsterflyingSquad[2] = { Dragon("Smaug"), HorseFly(123456) };

A reference to the objects must be passed

* 1. Monster \*monster = newFlyingMonster("Giant Bat", 100);

**Question 5**

Suppose *Printer* is a template class declared in a file called *printer.h* as follows:



\*\*\*Note that the constructer is not defined inline.

1. Write the entire printer.cpp file, which should contain the definition of *Printer*’s constructor. The constructor should make a **deep copy** of the value passed by parameter. You can assume that any **T** used will have any constructors or operators implemented that you need to initialize the value.

#include <cassert>

#include <iostream>

#include "stdafx.h"

template <class T>

class Printer

{

public:

Printer() : value(NULL) {}

Printer(T const\* value) : value(value) {}

Printer(T const& value) : value(new T(value)) {}

~Printer() { delete value; }

};

1. Read the information below and answer the questions that follow.

You are asked to create a class template called *TimeStampedPrinter*, which:

* Inherits publicly from *Printer*.
* Has two type parameters: One for the type of message to print, and one for the timestamp at which an object was created. You can assume that the user will choose an appropriate type for the timestamp.
* Has, in addition to the inherited members, a member variable called *timestamp* of the appropriate generic type.
* Has a constructor that takes two parameters: The first is used to initialize the message, and the second is used to initialize the timestamp.
* Has an overloaded *print* function that prints the timestamp, followed by a colon, followed by the message, followed by a newline.

Questions:

1. Write the class declaration. Do not write any inline function definitions.

template <class T>

class TimeStampedPrinter : public Printer<T>

{

private:

T messageType;

T timestamp;

public:

TimeStampedPrinter(T messageIn, T timestampIn);

void print();

};

1. Write the definition of the constructor.

TimeStampedPrinter(T messageIn, T timestampIn)

{

messageType = typeid(messageIn).name();

message = messageIn;

timestamp = timestampIn;

}

1. Write the definition of the *print* function.

void print()

{

timestamp << ";" << message << endl;

}