Quiz #2

Name: Solutions_____

Student Number: _____

Signature: _____

Instructions

1. **Fill in your Name, Student Number, and signature above.**
2. **This is a closed book Quiz. No electronic or paper aids permitted.**
3. **Do not open this test booklet until instructed to do so.**
4. **Clearly indicate if some part of your work is not to be marked. Add as many comments as needed to provide a clear response.**
5. **You may answer the questions in any order you want.**
6. **Raise your hand if you have a question. The instructor will come over to assist you.**
7. **Copying from or communicating with a neighbor or with anyone directly or electronically will result in both students receiving a zero and may result in further disciplinary action by the school and or university administration.**
8. **The total number of points for this Quiz is 50.**
9. **You may use the attached Operator Precedence chart and Syntax chart**
10. **You will have 30 minutes to complete this Quiz.**
11. **When you are finished, bring your paper and student card to the front of the room where you will hand in your quiz.**

Good luck!

Total = _____ / 50

Question	Max Mark	Actual Mark
1	10	
2	5	
3	5	
4	10	
5	10	
6	10	
Total	50	

1. Class Definition Questions – Multiple Choice 10 Marks

CHOOSE 1 OR 2 CORRECT OPTIONS PER QUESTION, not more.

(a) What does the term Stride refer to when discussing arrays?

1) the number of elements in the array
2) the index of the last element in the array
3) the number of bytes used to store an array
4) the number of bytes used to store an element of an array
5) the number of bytes used to store the pointer to the array

(b) The main benefit(s) of encapsulation is/are:

1) reduces system complexity
2) it eliminates the need for inter-module cohesion
3) allows friend functions to access class data via setters and getters
4) makes classes more difficult to program
5) hides implementation details

(c) Benefits of Inheritance include:

1) Reduces encapsulation of the base class and its derived classes
2) Enables software to be written in less time by re-using tested designs
3) Enables a child class method to be overridden by a base class
4) Enables classes to re-use existing methods and data of a base class
5) Enables multiple methods to be overloaded in the class

(d) Method Overloading occurs when:

1) More than one class implement the same method with the same signature
2) A method is invoked too often
3) A class declares a method multiple times with different signatures
4) A method is used to implement more than one thing
5) A method has the same name and parameters but returns different types

(e) Method Overriding occurs when:

1) More than one class implement a method but with different signatures
2) A child method has the same signature as its parent's virtual method
3) A class declares a method multiple times with different signatures
4) A method is used to implement more than one thing
5) A method has the same name and parameters but returns different types

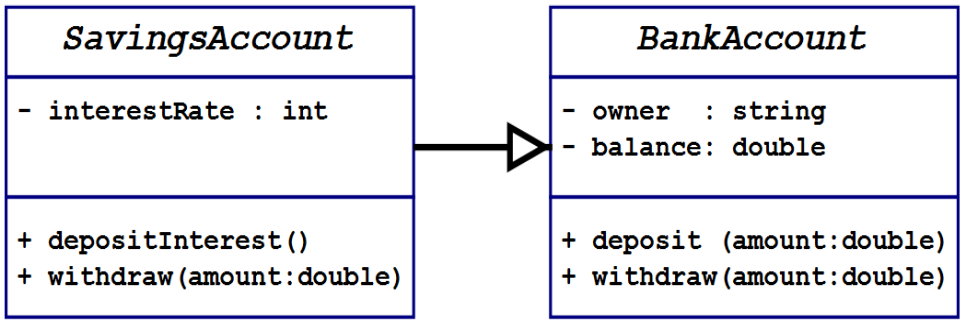
For questions with 2 correct answers:
- 2 Marks for getting both correct
- 1 Mark for getting one correct
- 0 Marks for none correct or more than 2 selected

For questions with 1 correct answer:
- 2 Marks for getting the correct answer
- 1 Mark for getting the correct answer plus one wrong answer
- 0 Marks for none correct or more than 2 selected

2. **Classes that incorporate dynamic memory can have problems unless "The Big Three" are implemented. Identify "The Big Three". 5 Marks**

- ☐ **The default constructor**
- ☐ **The delete operator**
- ☑ **The copy constructor**
- ☑ **The assignment operator**
- ☑ **The destructor**

3. **Complete the class declaration for the SavingsAccount class as depicted in the UML diagram below. Ensure that the class declaration includes a default constructor and captures any parent/child relationship. 5 Marks**




```
class SavingsAccount : public BankAccount {
public:  // Public  member declarations go here
   SavingsAccount();
   void depositInterest();
   void withdraw(double amount);
private: // Private member declarations go here
  int interestRate;
};
```

4. Dynamic Arrays as Return Values - helloName **10 Marks**

Given a C string parameter called `name`, e.g. "Bob", return a greeting of the form "Hello Bob!". Your function will accept a nullptr or a C string that is null terminated, and must return a new dynamic char array that is also null terminated.

Your function must have the following signature:

```
char* helloName(const char name[]);
```

For example:

```
helloName("Bob")    → "Hello Bob!"
helloName("Alice") → "Hello Alice!"
helloName("X")      → "Hello X!"
helloName(nullptr) → "Hello !"
```

```
char* helloName(const char name[]) {
    // set len=0 in case name equals nullptr
    int len = 0;

    // don't call strlen if name==nullptr
    if (name != nullptr)
        len = strlen(name);

    // Allocate enough chars for retStr
    char *retStr = new char[len+8];

    // Use strcpy because retstr is not yet
    // initialized. strcat may fail if used.
    strcpy(retStr, "Hello ");

    // cat name if it was not nullptr
    if (name != nullptr)
        strcat(retStr, name);

    // Complete answer by adding the "!" string
    strcat(retStr, "!");

    // return completed retStr to caller
    return retStr;
}
```

Alternative solution using pointer arithmetic:

```
char* helloName(const char name[]) {
    // set len=0 in case name equals nullptr
    int len = 0;

    // don't call strlen if name==nullptr
    if (name != nullptr)
        len = strlen(name);

    // Allocate enough chars for retStr
    char *retStr = new char[len+8];

    // Use strcpy because retstr is not yet
    // initialized. strcat may fail if used.
    strcpy(retStr, "Hello ");

    // cat name if it was not nullptr
    if (name != nullptr)
        strcpy(retStr+6, name);

    // Complete answer by adding the "!" string
    strcpy(retStr+len+6, "!");

    // return completed retStr to caller
    return retStr;
}
```

5. Working with Two-Dimensional Arrays **10 Marks**

Write a void function called addToColumn that accepts a two-dimensional int arr[ ][5] array, the number of rows in that array, a column number, and an int value that is to be added to every element in that column.

Your function must have the following signature:

```
void addToColumn(int arr[][5], int numRows, int colNum, int numToAdd);
```

For example:

Given int A[5][5] where:

```
A = {{1,0,0,0,0},
     {0,1,0,0,0},
     {0,0,1,0,0},
     {0,0,0,1,0},
     {0,0,0,0,1}}
```

After calling addToColumn:

```
addToColumn(A, 5, 2, 1);
```

Array A will contain:

```
A = {{1,0,1,0,0},
     {0,1,1,0,0},
     {0,0,2,0,0},
     {0,0,1,1,0},
     {0,0,1,0,1}}
```

```
void addToColumn(int arr[][5],int numRows,int colNum,int numToAdd){

// check for bad input before starting
// ***********************************************
// For Quiz, no marks lost if did not do check
// ***********************************************
if (arr != nullptr)
   if (numRows > 0)
       if ((colNum >= 0) && (colNum < 5))

          // Valid input
          // Loop through each row adding numToAdd
          // ************************************
          for(int row=0; row<numRows; row++)
              arr[row][colNum] += numToAdd;
```

6. Sorting an Array of Doubles **10 Marks**

Write a void function called doubleSort that accepts an array of doubles `d` and the number of elements in that array `num`. This function should sort the elements such that:

$$d[0] \leq d[1] \leq \ldots \leq d[num-1]$$

For example:
Given array D[7] where:

```
D = {6.6, 5.5, 4.4, 3.3, 2.2, 1.1, 0}
```

After calling:

```
doubleSort(D, 7);
```

Array D will contain:

```
D = {0, 1.1, 2.2, 3.3, 4.4, 5.5, 6.6}
```

```
void doubleSort(double d[], int num) {
// Any sort will do, but this is a bubbleSort
// ******************************************
// This FOR loop looks to "bubble up" the
// largest double from d[0] to d[i] to d[i].
// i starts with i=num-1, so first pass finds
// the largest number in array and places it
// in d[num-1]. Next loop finds next largest
// and places it in d[num-2], and so on.
for(int i=num-1; i>0; i--)

    // This FOR searches for the largest of
    // those remaining to be bubbled into d[i].
    for(int j=0; j<i; j++)

        // Check if numbers are out of order
        // Swap them if they are.
        if (d[j] > d[j+1]) {
            // d[j] and d[j+1] in wrong order
            // So let's swap them!
            // ***************************
            double tmp = d[j];
            d[j]       = d[j+1];
            d[j+1]     = tmp;
        }
}
```

C++ Operator Precedence – Appendix 2

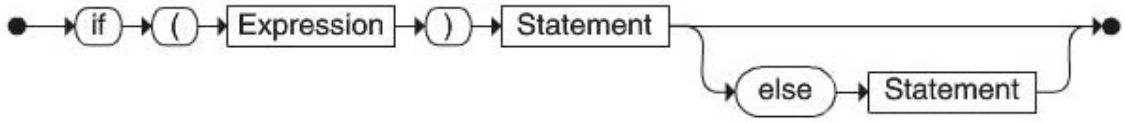
:: scope resolution operator
. dot operator -> member selection [] array indexing ( ) function call ++ postfix increment operator (placed after the variable) -- postfix decrement operator (placed after the variable)
++ prefix increment operator (placed before the variable) -- prefix decrement operator (placed before the variable) ! not - unary minus + unary plus * dereference & address of *new* *delete* *delete[]* *sizeof*
* multiplication / division % remainder (modulo)
+ addition - subtraction
<< insertion operator (output) >> extraction operator (input)
< less than <= less than or equal > greater than >= greater than or equal
== equal != not equal
&& and
|| or
= assignment += add and assign -= subtract and assign *= multiply and assign /= divide and assign %= modulo and assign

Highest precedence (done first)

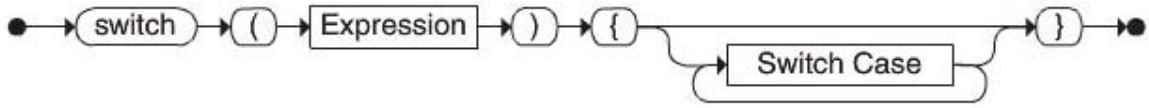
Lowest precedence (done last)

C++ Flow of Control Statement Syntax

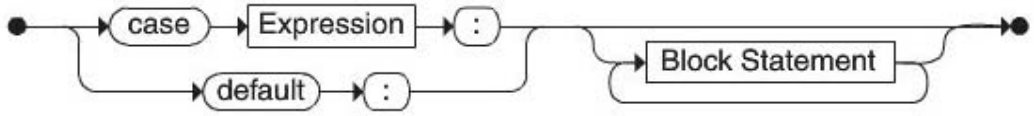
If Statement



Switch Statement



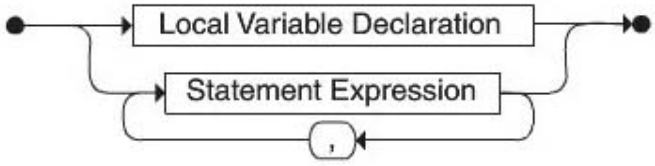
Switch Case



While Statement

For Statement

For Init



For Update