C++ AP Exam Review Name -

A. Expressions, Types & Operators
- int (4 bytes), double (8 bytes), bool, char (1 byte)
- enum DaysOfWeek {Sun = 1, Mon, Tue, Wed, Thu, Fri};
- order of operations % (mod) is tie with multipl. & division; && (AND) is before || (OR)
- integer division 5/2 = 2
- promotion 5.0/2 = 2.5 due to implicit promotion
- typecasting double(5)/ 2 = 2.5 but double (5/2) = 2 due to order of operations
- int j, k, n; is legal as is int j = 0, k = 2, n = 10; and j = k = n = 0;

B. Statements
- if one line if statements are legal such as if (n > 10) cout << "hello";
- switch break; statements are NOT required in each case of a switch structure

```
switch (result)
{
  case 1: case 2: case 3:
    cout << "1, 2, or 3" << endl;
  case 4:
    cout << "4" << endl;
    break;
  default:
    cout << "none of the above" << endl;
}
```

- while loops & do/while loops are interchangeable however do/while loops guarantee one iteration
- for loops are interchangeable with while loops

```
for (int row = 0, int col = 0; row < 3 && col < 3; row++, col++) myMatrix[row][col] = 7;
```

is a legal one-line for loop.

C. Functions
- Pass by reference when you want the function to change the value of the corresponding argument
- Pass by reference when you want to avoid the memory overhead of copying a large argument (strings, vectors, matrices, etc.)
- Pass by constant reference if you want to avoid the memory overhead but want to protect yourself from accidentally changing an actual parameter.
- every recursive function must have a base case, otherwise stack overflow is likely to occur
- (AB only) tail recursive functions do not have executable statements that follow the recursive function call; tail recursive functions decrease the amount of stack overhead and are more efficient
- two functions are overloaded if they have the same exact name but different parameter lists, where one function has more parameters than the other or at least one parameter is of a different data type

D. Structs and Classes
- An abstract data type is a general set of specifications of what a user-defined data type should be able to do.
- A struct is equivalent to a class except that all members are public by default. In a class, all members are private by default. Usually, we use struct definitions when we simply want to store multiple pieces of information of different data types in one variable. We use classes when we want to add functionality (i.e. member functions) to a user-defined data type. Remember that structs can have constructors and member functions just like classes.
- Use initializer lists when writing constructors

```
Point::Point(): myX(0), myY(0)
```

rather than

```
Point::Point()
{
  myX = 0;
  myY = 0;
}
```

E. AP Classes

- Memorize the existence and use of the member functions of apstring , apvector, & apmatrix since you will not have a copy of these class header files on the Part I Multiple Choice section. AB students must memorize the member functions of apstack & apqueue as well.

- apstring - length, substr, find, [], +, +=, getline

```
cout << myName.length();
cout << myName.substr(myName, 1);
if (myName.find('a')) cout << "An a is found in my name";
cout << lastName + firstName;
firstName += lastName;
getline(infile, myName);
```

- apvector - constructors, length, resize

```
apvector <int> scores(10, 0);          // where scores has length of 10 and fill value of 0
```

- apmatrix - numrows, numcols, resize

- (AB only) apstack - push, pop, pop(itemType &), makeEmpty, top, isEmpty, length
- (AB only) apqueue - enqueue, dequeue, dequeue(itemType &), makeEmpty, front, isEmpty, length

F. (AB only) Big-O

- see charts in textbook

G. Sorting and Searching

- Selection sort, insertion sort, quick sort, merge sort
- sequential search, binary search
- (AB only) hashing

H. (AB only) Linked Lists

- singly-linked, doubly-linked, circular
- typical node definition

```
struct Node
{
  int data;
  Node *next;
};
```

- using new and delete operators
- inserting and deleting nodes to a linked list

I. (AB only) Binary Trees

- typical node definition

```
struct TreeNode
{
  int data;
  TreeNode *left;
  TreeNode *right;
};
```

- preorder, postorder, inorder traversals of trees
- differences between binary tree, binary search tree, binary expression tree, & heaps