CSC 210 (Spring, 1999) Sample Final Exam Key page 1 of 4

Note: For questions related to linked lists, assume the class List has a data member my_head and that class Node is defined as follows:

```cpp
class Node {
    public:
        int my_data;
        Node * my_link;
    Node( int data, Node * link = 0 ){
        my_data( data ), my_link( link ) {} 
    ~Node() { delete my_link; }
};
```

General Programming Skills.

For each of the following standard programming situations, write syntactically correct C++ code to accomplish the required task.

1. [5 points] Declare a pointer variable A and make it point to an allocated array of 100 integers that has all 0’s in it.

```cpp
    int SIZE = 100;
    int * A = new int[ SIZE ];
    for ( int i = 0; i < SIZE; ++i ) A[ i ] = 0;
```

2. [5 points] Make the variable s_new point to a freshly allocated C-string that is a copy of the string pointed to by s_old (assume both variables have been declared; you may use anything in string.h except strcpy).

```cpp
    char * s_new = new char[ strlen( s_old ) + 1 ];
    strcpy( s_new, s_old );
```

3. [5 points] Add the integer stored in the_int (previously declared) to the front of a linked list pointed to by my_head (just write the actual code, not the member function).

```cpp
    my_head = new Node( the_int, my_head );
```

4. [5 points] Remove and deallocate the first node of a linked list pointed to by my_head.

```cpp
    Node * trash = my_head;
    my_head = my_head->my_link;
    trash->my_link = 0;
    delete trash;
```

5. [5 points] Read characters from standard input until a Q is encountered and store all (except the Q) as a C-string in an array buffer (assume buffer has been declared to hold 512 bytes; your code should not write beyond the end of the array; either truncate the input or abort instead).

```cpp
    char ch; int index = 0;
    cin.get( ch );
    while ( ’Q’ != ch && index < 511 ){
        buffer[ index++ ] = ch;
        cin.get( ch );
    }
    buffer[ index ] = ’\0’;
```
Linked List Programming.

Each of these two problems should be solved without using any helper functions.

6. [12 points] Write a recursive function `int min(Node * p)` that returns the minimum integer in the linked list pointed to by p. Your function should return `INT_MAX` (the maximum possible value for an `int` — assume this is already defined) if p is null.

```c
int min(Node * p)
{
    int retval = INT_MAX;
    if (p)
    {
        retval = min(p->my_link);
        if (p->my_info < retval) retval = p->my_info;
    }
    return retval;
}
```

7. [18 points] Write an iterative member function `swap_last_two` for the `List` class that reverses the order of the last two nodes of the list. Your function should not access the `my_data` field of any node. If the list has less than two nodes, it should do nothing.

```c
void List::swap_last_two()
{
    Node * previous = my_head;
    if (0 == previous) return;
    Node * first = previous->my_link;
    if (0 == first) return;
    Node * second = first->my_link;
    if (0 == second) {
        my_head = first;
        first->my_link = previous;
        previous->my_link = 0;
        return;
    }
    while (second->my_link) {
        previous = first;
        first = second;
        second = first->my_link;
    }
    previous->my_link = second;
    second->my_link = first;
    first->my_link = 0;
}```
Abstract Classes, Inheritance, and Virtual Functions.

8. [25 points] What does the following program print?

```c++
#include<iostream.h>
#include<string.h>

class Base {
  public:
    Base( const char * name )
    { cout << "Base::Base, " << name << endl;
      my_name = new char[ strlen( name ) + 1 ];
      strcpy( my_name, name ); }
    void beta() { cout << "Base::beta, " << endl; }
    virtual void gamma() = 0;
    virtual "Base"()
    { cout << "Base::"Base, " << my_name << endl;
      delete [] my_name; }
  protected:
    char * my_name;
};

class Zero: public Base {
  public:
    Zero( const char * name, int number ):
      Base( name ), my_number( number )
    { cout << "Zero::Zero, " << number << endl; }
    void beta() { cout << "Zero::beta, " << endl; }
    virtual void gamma() { cout << "Zero::gamma, " << endl; }
    virtual "Zero"()
    { cout << "Zero::"Zero, " << my_number << endl; }
  protected:
    int my_number;
};

class One: public Zero {
  public:
    One( const char * name, int number,
         const char * occupation ):
      Zero( name, number )
    { cout << "One::One, " << occupation << endl;
      my_occupation = new char[ strlen( occupation ) + 1 ];
      strcpy( my_occupation, occupation ); }
    void beta() { cout << "One::beta, " << endl; }
    virtual void gamma() { cout << "One::gamma, " << endl; }
    virtual "One"()
    { cout << "One::"One, " << my_occupation << endl;
      delete [] my_occupation; }
  protected:
    char * my_occupation;
};

int main()
{
  Zero * p0 = new Zero("Nix", 0);
  One * p1 = new One("Me", 1, "Prof");
  p0->beta(); p0->gamma();
  p1->beta(); p1->gamma();
  Base * B0 = p0; Base * B1 = p1;
  B0->beta(); B0->gamma();
  B1->beta(); B1->gamma();
  delete B0; delete B1;
}
```

```
Base::Base, Nix  Base::beta
Zero::Zero, 0  Zero::gamma
Base::Base, Me  Base::beta
Zero::Zero, 1  One::gamma
One::One, Prof  Zero::"Zero, 0
Zero::beta  Base::"Base, Nix
Zero::gamma  One::"One, Prof
One::beta  Zero::"Zero, 1
One::gamma  Base::"Base, Me
```
Finite-State Machines.

9. [10 points] Design a finite-state machine for a program that reads input and prints all HTML commands, one command per line. For the purposes of this exercise, an HTML command is any string that appears between angle brackets <>. For example, if the input is

```
Here is some <strong>bold</strong>
face</strong> text,
and here is some <em>text</em> in italics</em>.
```

Do not write the actual program, but indicate the actions of the program on the transitions of the FSM.

![Finite-State Machine Diagram]

**cout << endl**

**EXTRA CREDIT.**

10. [10 points] What are the phases of the “software life cycle”? Explain briefly what happens during each phase.

1. **Requirements Analysis.** Why is this software being written? What needs should it address?

2. **Specification (User-Level Design).** What precisely does the software do from a user’s point of view? How does it respond to each possible input from the user?

3. **System Design.** What are the main components of this software and how do they communicate with each other?

4. **Detailed Design.** How will the design be rendered in the chosen programming language? In C++, this means figuring out what the classes and methods will be and writing header files for them.

5. **Coding.** The actual writing of the program.

6. **Validation and Verification.** Does this software serve its intended purpose? This “phase” should be an integral part of all the others and needs to involve both testing (subjecting design or code to both typical and atypical scenarios) and logical analysis.