Name: KEY

Score ___105__ / 100

1. (5 Points) Place the following variables in order by the largest number they could store. There is no need to actually list the largest storable value. Indicate the relationship between the types using <, >, <=, >=, or ==.

```
long l;
signed char k;
char c;
int i;
unsigned long u;
short m;
```

k < c <= m <= i <= l < u

2. (10 Points) Presented below is a Makefile being used to compile a simple project. As work on the project continues, two new files are created: magic.cpp and magic.h. Assuming the files are correctly formatted, and magic.h is included in both magic.cpp and prog5.cpp, update the Makefile to perform *both compilation and dependency checking* on the new files.

```
CPP = g++
FLAGS = -g -Wall

EXEC = prog5
OBJS = prog5.o zyphon.o magic.o

default:${EXEC}

clean:
        rm -f ${EXEC}
        rm -f *.o

run: ${EXEC}
        ./${EXEC}

${EXEC}:${OBJS}
        ${CPP} ${FLAGS} -o ${EXEC} ${OBJS}

.cpp.o:
        ${CPP} ${FLAGS} -c $<

zyphon.o: zyphon.cpp zyphon.h
prog5.o: prog5.cpp zyphon.h magic.h
magic.o: magic.cpp magic.h
```

3. Each of the following code sections have some errors in them. Locate the errors and fix them.

(a) (5 points)

```
float ptr*;
```

```
float *ptr;
```

(b) (5 points)

```
char pi;
int floptr = &pi;
```

```
char pi;
char * floptr = &pi;
```

(c) (5 points)

```
int * intprt = &intval;
int intval = 5;
```

```
int intval = 5;
int * intprt = &intval;
```

(d) (5 points) Assuming class foobar is defined

```
foobar myfoo = new foofar();
```

```
foobar * myfoo = new foobar();
```

4. (5 points) There are four important differences between a reference and a pointer. Name (at least) two of them.

A reference cannot be null

Once established, a reference cannot be changed

It does not need to be dereferenced

All operators operate on the referenced value

5. (10 points) Write a section of C++ code that reserves a memory location for storing 15 integer values from *heap memory*, populates the array with the values 1 to 15, uses `cout` to print out the contents of the array, and finally releases the memory for later use. There is no need to write a complete function, and you may assume all of the necessary headers are already included.

```
int *array = new int[15];
for(int i = 0; i < 15; i++) {
      array[i] = i + 1;
}
for(int j = 0; j < 15; j++) {
      cout << array[j] << " ";
}
cout << endl;
delete[] array;
```

6. (20 points) Below is the UML diagram for a class. Write a class header for the described class. Assume all of the required data types have already been defined. There is no need to write accessor methods.

ParticleEmitter
-LocX : int -LocY : int -Angle : double -Type : Particle
+ParticleEmitter() +ParticleEmitter(x : int, y : int) +Emit() : Particle -ExamineParticle() : void

```
Class ParticleEmitter {
      public:
            ParticleEmitter();
            ParticleEmitter(int x, int y);
            Particle Emit();
      private:
            int LocX;
            int LocY;
            double Angle;
            Particle Type;
            void ExamineParticle();
};
```

7. (5 *bonus* points) What purpose is served by the following code snippet? Be specific. You may assume normal C/C++ string semantics hold.

```
char string1[100];
char string2[100];

/* stuff happens */

char *p, *q;
p = string2;
q = string1;
while(*p++ = *q++)
      ;
```

Copies the contents of string1 into string 2

8. (15 points) Consider the following program. What output will it produce?

```
#include <iostream>

using namespace std;

class Foo {
    public:
        virtual int isA() { return 0; }
        int looksLike() {return 0; }
};

class Bar : public Foo {
    public:
        virtual int isA() { return 1; }
        int looksLike() { return 1; }
};

int main () {
    Foo *foo = new Foo();
    Bar *bar = new Bar();

    Foo * ptr1 = foo;
    Foo * ptr2 = bar;

    cout << "foo is a: " << foo->isA() << endl;
    cout << "foo looks like: " << foo->looksLike() << endl;
    cout << "bar is a: " << bar->isA() << endl;
    cout << "bar looks like: " << bar->looksLike() << endl;
    cout << "ptr1 is a: " << ptr1->isA() << endl;
    cout << "ptr1 looks like: " << ptr1->looksLike() << endl;
    cout << "ptr2 is a: " << ptr2->isA() << endl;
    cout << "ptr2 looks like: " << ptr2->looksLike() << endl;
    return 0;
}
```

```
foo is a: 0
foo looks like: 0
bar is a: 1
bar looks like: 1
ptr1 is a: 0
ptr1 looks like: 0
ptr2 is a: 1
ptr2 looks like: 0
```

9. Consider the following class...

```
class Stack {
    private:
        int *array;
        int ptr;
        int size;

    public:
        Stack();
        Stack(int s);
        Stack(const Stack & s);
        ~Stack();
        int push (int item);
        int pop(int & item);
};

Stack::Stack() {
    ptr = 0;
    size = 10;
    array = new int[size];
}

Stack::Stack(int s) {
    ptr = 0;
    size = s;
    array = new int[size];
}

Stack::~Stack() {
    delete array;
}

int Stack::push(int item) {
    if(ptr == size) {
        int *newbie;
        newbie = new int[size * 2];
        if (newbie == 0) {
            return 0;
        }
        for(int i = 0; i < size; i++) {
            newbie[i] = array[i];
        }
        size = 2 * size;
        delete array;
        array = newbie;
    }
    array[ptr] = item;
    ptr++;
    return 1;
}
```

(15 points) Write an implementation of the copy constructor.

```
Stack::Stack(const Stack & s) {
      size = s.size;
      ptr = s.ptr;
      array = new int[size];

      for(int i = 0; i < ptr; i++) {
            array[i] = s.array[i];
      }
}
```