Programming Design, Spring 2015

Suggested Solution for Midterm Exam

Problem 1 (Solution provider: Shelley Sun)

(a) True.

(b) True.

(c) True. while(condition){do_thing();} -> for(;condition;){do_thing();}

(d) False. We can use the scope resolution operator :: to access the global variable.

(e) True.

(f) True.

(g) True.

(h) False. a[0][0] is the first element but &a[0][5] is the address of the 6th element. &a[0][5] - a[0][0] will not be 5

(i) True.

(j) False. Just like usual functions, a constructor may have a default argument.

Problem 2 (Solution provider: Shelley Sun)

(a)

When we drop { } of if-else, our programs may be grammatically ambiguous. C++ defines that "one else will be paired to the closest if that has not been paired with an else."

- Never drop { }.
- Drop { } only when you know what you are doing.

(b)

The reference is another variable that refers to the variable. Thus, using the reference is the same as using the variable.

Instead of declaring a usual local variable as a parameter, declare a reference variable. This is to "call by reference".

Thus we can call by reference and modify our parameters' value.

(c)

If a variable is actually a symbolic constant.

If we want to prevent a variable from being modified.

(d)

- The initializer can be called automatically.
- We can set visibility of members in a class, so we can hide/open our instance members.
- By data hiding, we control the way that members are accessed. We can better predict how others may use our class.
- Spaces allocated dynamically will be released automatically.

Problem 3 (Solution provider: Willy Liao)

```
int result=0; //store the result
for(int i=0; i<n; i++) {
        for(int j=i+1; j<n; j++) { //start from i+1 to prevent double counting
              if(mat[i][j]!=mat[j][i]) // check if they are different
                        result++;
        }
}
cout<<result<<endl; //print out
```

Problem 4 (Solution provider: Willy Liao)

(a)

29 84 5 2 0 0 0 0 0 0

(b)

Assume that the algorithm start from a seed that smaller than 100. It first computes the square of the seed, then chooses the third digit and the second digit as the output random number and the new seed.

It's bad because when the number becomes small, the output will finally converge to 0.

(c) No, because the parameter s in the srand is declare by unsigned int. s in the srand is always positive.

(d)

Original:

```
for(int i = 3; i >= 0; i--)
{
        int digit = floor(curNum / pow(10, i));
        next[3 - i] = digit;
        curNum -= digit * pow(10, i);
}
```

After modified:

```
int i=3;
while(i>=0)
{
        int digit = floor(curNum / pow(10, i));
        next[3 - i] = digit;
        curNum -= digit * pow(10, i);
        i--;
}
```

(e)

The storage of the two integers can be improved. The struct can use one character to store the value(“char cur;”). The functions srand and rand should be modified as below.

```
bool Randomizer::srand(unsigned int s)
{
        if(s < 100)
        {
            cur=s;
            return true;
        }
        else
                return false;
}
int Randomizer::rand()
{
    int curNum = cur;
    curNum = curNum * curNum;
    int next[4] = {0};
    for(int i = 3; i >= 0; i--)
    {
        int digit = floor(curNum / pow(10, i));
        next[3 - i] = digit;
        curNum -= digit * pow(10, i);
    }
    cur=next[1]*10+next[2];
    return cur;
}
```

(f)

For this class, we need constructor to prevent user from calling rand() before cur[2] is initialized. We need neither copy constructor nor destructor since the cur array is not dynamic.

Problem 5 (Solution provider: Tammy Chang)

(a)

The correct constructors and destructor should be like:

```
//constructor
Person::Person(int id) : id(id){
        n = 0;
        m = NULL;
}
Person::Person(int id, int n, int* m) : id(id){
        this->n = n;
        this->m = new int[n];
        for(int i=0; i<n; i++){
                this->m[i] = m[i];
        }
}
```

id is a constant variable, we need a member initializer to initialize it. Moreover, we don't want the m array in Person contain the same address as the input array, we should dynamically allocate a space and copy the values to it.

```
//destructor
Person::~Person(){
        delete[] m;
        m = NULL;
}
```

(b)

```
void Person::knowPerson(int anId){
        n++;
        int* temp = new int[n];
        for(int i=0; i<n-1; i++)
                temp[i] = m[i];
        temp[n-1] = anId;
        delete[] m;
        m = temp;
}
```

We copy the values in m to temp and add a new ID in the last position. We need to delete the original array m first and let m point to the same address as temp.

(c)

We see a compilation error because the non-constant instance function can't be called by a constant object. Therefore, we should change the print function in our class into `void print() const;`

(d)

We see a run time error because we use call-by-value method with default copy constructor, which only does shallow copy. When p is passed to globalPrint2 function, the copy constructor copies p and let p.m (inside the globalPrint2 function) points to the same address as the p in the main function. After the globalPrint2 function is ended, a destructor is called and p (inside the globalPrint2 function) is deleted, which deleted the m array pointed by p (in the main function). Therefore, we should write a copy constructor that does deep copy.

```
//copy constructor
Person::Person(const Person& p) : id(p.id){
        n = p.n;
        m = new int[n];
        for(int i=0; i<n; i++){
                m[i] = p.m[i];
        }
}
```