

CSCI 135 Software Design and Analysis I

Homework 4

Solution

Saad Mneimneh
Visiting Professor
Hunter College of CUNY

Problem 1: Intervals

We would like to create a new type to provide the ability to manipulate inexact quantities. Such quantities can be represented by intervals of the form $[a, b]$ with $a \leq b$.

(a) Define a class Interval that contains the necessary member data to represent the low and high ends of an interval, in addition to a constructor to construct interval objects. The class should allow you to write the following program:

```
int main() {
    Interval r=Interval(6.12,7.48); //construct the interval [6.12,7.48];
}
```

ANSWER:

```
float min(float x, float y) {
    if (x<y)
        return x;
    return y;
}
```

```
float max(float x, float y) {
    if (x>y)
        return x;
    return y;
}
```

```

class Interval {
public:
    float l;
    float h;

    Interval(float l, float h) {
        this->l=min(l,h);
        this->h=max(l,h);
    }
};

```

(b) Define two functions outside the class to extract the low and high ends of an interval. Note that the member data of class Interval must be public for this to work. Your functions must allow you to write the following program:

```

int main() {
    Interval r=Interval(6.12,7.48);
    cout<<' '[ '<<low(r)<<' ','<<high(r)<<''] '<<' '\n' ';
}

```

ANSWER:

```

float low(Interval r) {
    return r.l; //ok, l is public in class Interval
}

float high(Interval r) {
    return r.h; //ok, h is public in class Interval
}

```

(c) Move the above two functions inside the class. Make any changes that are necessary, including making the member data of class Interval private. Write a small program to illustrate how your class works.

ANSWER:

```

class Interval {
private:
    float l;
    float h;

public:
    Interval(float l, float h) {
        this->l=min(l,h);
        this->h=max(l,h);
    }

    float low() {
        return l;
    }
}

```

```

float high() {
    return h;
}
};

int main() {
    Interval r=Interval(6.12,7.48);
    cout<<' ' <<r.low()<<' ' <<r.high()<<' ' <<' '\n';
}

```

(d) Add a default constructor that constructs the interval [0,1].

ANSWER:

```

class Interval {
    float l;
    float h;

public:
    Interval(float l, float h) {
        this->l=min(l,h);
        this->h=max(l,h);
    }

    Interval(float c, int t) {
        float delta=c*(float)t/100;;
        l=min(c-delta,c+delta);
        h=max(c-delta,c+delta);
    }

    Interval() {
        l=0;
        h=1;
    }
}

```

(e) Add a member function to set the low and high ends of an interval.

ANSWER:

```

class Interval {
    float l;
    float h;

public:
    Interval(float l, float h) {
        set(l,h);
    }

    Interval(float c, int t) {
        float delta=c*(float)t/100;;
        set(c-delta,c+delta);
    }
}

```

```

Interval() {
    set(0,1);
}

void set(float l, float h) {
    this->l=min(l,h);
    this->h=max(l,h);
}

float low() {
    return l;
}

float high() {
    return h;
}
};

```

(f) Add a member function to check whether a number is contained in an interval. For instance, if your member function is called `contains(...)`, then you should be able to use it as follows:

```

int main() {
    Interval r=Interval(6.12,7.48);
    if (r.contains(6.5)) {
        //do something
    }
}

```

ANSWER:

```

class Interval {
    float l;
    float h;

public:
    Interval(float l, float h) {
        set(l,h);
    }

    Interval(float c, int t) {
        float delta=c*(float)t/100;;
        set(c-delta,c+delta);
    }

    Interval() {
        set(0,1);
    }
}

```

```

void set(float l, float h) {
    this->l=min(l,h);
    this->h=max(l,h);
}

float low() {
    return l;
}

float high() {
    return h;
}

bool contains(float x) {
    return (x>=l && x<=h);
}
};

```

(g) Consider the following operator overloading (outside the class):

```

Interval operator+(Interval r1, Interval r2) {
    return Interval(r1.low()+r2.low(), r1.high()+r2.high());
}

```

assuming that low() and high() are the member functions of part (c).

Define analogous functions to subtract two intervals, multiply two intervals, and divide two intervals.

$$[a, b] - [c, d] = [a - d, b - c]$$

$$[a, b] \cdot [c, d] = [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)]$$

$$[a, b]/[c, d] = [a, b] \cdot [1/d, 1/c] \text{ if } 0 \notin [c, d]$$

ANSWER:

```

Interval operator-(Interval r1, Interval r2) {
    return addInterval(r1, Interval(-r2.high(), -r2.low()));
}

Interval operator*(Interval r1, Interval r2) {
    float a=r1.low()*r2.low();
    float b=r1.low()*r2.high();
    float c=r1.high()*r2.low();
    float d=r1.high()*r2.high();
    return Interval(min(a,min(b,min(c,d))),max(a,max(b,max(c,d))));
}

```

```
Interval operator/(Interval r1, Interval r2) {
    if (!r2.contains(0))
        return mulInterval(r1, Interval(1/r2.high(),1/r2.low()));
    else {
        cout<<"error: divide by 0\n";
        exit(0); //this will exit the program
    }
}
```

Problem 2: Arrays and shifts

(a) Write a function that accepts an array a and reverses it.

ANSWER:

```
//this will reverse the sub-array a[start]...a[end]
void reverse(int * a, int start, int end) {
    int i=start;
    int j=end;
    while (i<j) {
        char temp=a[i];
        a[i]=a[j];
        a[j]=temp;
        i++;
        j--;
    }
}
```

```
//this will reverse the entire array
void reverse(int * a, int n) {
    reverse(a,0,n-1);
}
```

(b) Write a function that accepts an array a and an integer s , and performs a right circular shift by s on the array. For instance, if the array is an array of characters representing the string "ababaca", a circular shift by 5 makes it "abacaab". Assume s is less than the length of the array. Do this with the following strategies:

- Use extra space: While copying elements from one part of the array to another, some elements will be stepped over and deleted. Use an extra array of size s as a scratch space to save those elements as you move things around.
- Use extra time: Perform a circular shift of 1, thus requiring extra space independent of s . But repeat the operation s times.
- No extra space or time: In part (a), we need extra space dependent on s . In part (b), we need to spend a time dependent on ns , where n is the length of the array, because we perform a circular shift s times. Try to perform the circular shift with both space and time independent of s . *Hint:* think in terms of part (a).

ANSWER: first approach//

```
//helper function
//swaps at most s elements between
//array and scratch space and returns
//number of swaps
int swap(int * scratch, int s, int * a, int n) {
    int count=0;
    for (int i=0; i<s && i<n; i++) {
        int temp=scratch[i];
        scratch[i]=a[i];
        a[i]=temp;
        count++;
    }
    return count; //number of swaps
}
```

```
void circularShift(int * a, int n, int s) {
    //decalre a scratch array dynamically
    //to hold the shifted part
    int * scratch=new int[s];

    int count=s; //number of swaps
    //as long as we make s swaps
    for (int i=0; count==s; i++)
        //a[i*s]...a[n-1] contains n-i*s elements
        count=swap(scratch, s, a+i*s, n-i*s);

    //fix first part
    for (int i=0; i<count; i++)
        a[i+s-count]=scratch[i];
    for (int i=count; i<s; i++)
        a[i-count]=scratch[i];

    //free memory
    delete[] scratch;
}
```

ANSWER: second approach//

```
void circularShift2(int * a, int n, int s) {
    for (int i=0; i<s; i++) //more time
        circularShift(a, n, 1); //less space
}
```

ANSWER: third appraoch//

```
//time is proportional to 2n, independent of s
//extra space is constant, independent of s or n
void circularShift3(int * a, int n, int s) {
    reverse(a, 0, s-1);
    reverse(a, s, n-1);
    reverse(a, 0, n-1);
}
```