CSCI 220 -- Homework 1

Nero Li

Chapter 1

R-1.11

```
class Flower
{
    private:
        string name;
        int pedal;
        float price;
    public:
        Flower(string name, int pedal, float price)
        : name(name), pedal(pedal), price(price) {}
        void setName(string str)
        {   name = str; }
        void setPedal(int n)
        {   pedal = n;  }
        void setPrice(float n)
        {   price = n;  }
        string getName()
        {   return name;    }
        int getPedal()
        {   return pedal;   }
        float getPrice()
        {   return price;   }
};
```

R-1.18

```
bool isMultiple(long n, long m)
{
    double i;
    i = (double)n / m;
    if (i - (long)i)
        return false;
    else
        return true;
}
```

R-1.20

```
int func(int n)
{
    return n * (n - 1) / 2;
}
```

C-1.5

- Create a bool array exist[52], and initalize them all to false.
- Create an integer array arr[52] containing queue after random.
- From i = 0 to 51:
    - Create an integer cur.
    - Use rand() to get a random number from 1 to 52, let it equal to cur.
    - If exist[cur – 1] is true, do previous step again.
    - arr[i] = cur.
- Return arr.

C-1.6

- Create an integer array character[6].
- From char[0] = 0 to 5
    - From char[1] = 0 to 5
        - From char[2] = 0 to 5
            - From char[3] = 0 to 5
                - From char[4] = 0 to 5
                    - From char[5] = 0 to 5
                        - Get six character by ‘a’ + each member in character array
                        - Combine all character together as a string and output the string.

Chapter 2

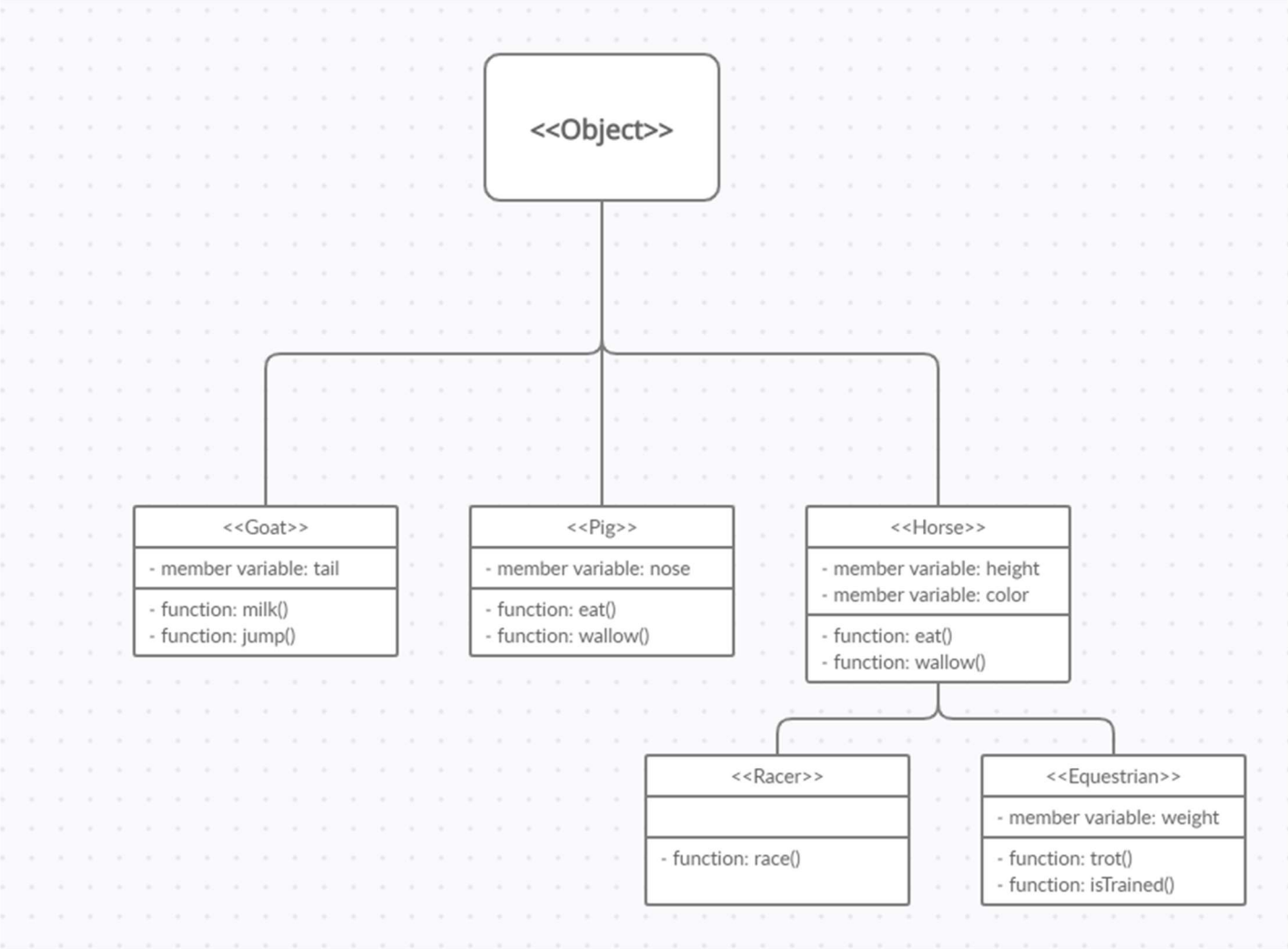
R-2.1

Have a very deep inheritance tree will make your program create several useless empty space if you create a new D class, since D class needs B, C, and A class during the construction. Also, compiler cannot quickly understand which function has been override to a new function because if you want to use a virtual function in class A, you will need to walk through a long trip from A to D.

R-2.2

Have a very shallow tree didn't use any advantages from the inheritance since inheritance is something that help you sort your data and then make your program methodical. If everything is extended from one class, it is almost equal to using no inheritance in this program.

R-2.6



C-2.4

```
class Line
{
    private:
        double a;
        double b;
    public:
        Line(int a, int b)
        : a(a), b(b) {}
        double intersect(Line l)
        {
            double result;

            try
            {
```

```
                if (a == l.a)
                {
                    throw Parallel("The two lines are parallel.");
                }
                else
                {
                    result = (l.b - b) / (a - l.a);
                }
            }
            catch(Parallel& err)
            {
                std::cerr << err.what() << '\n';
                exit(-1);
            }

            return result;
        }
};
```

Chapter 3

R-3.6

- Create a new member variable integer size and initialize it to 0.
- When insert a new node to linked list, ++size before the function return.
- When remove a node to linked list, --size before the function return.
- Add a function size() and return the member variable size.

R-3.11

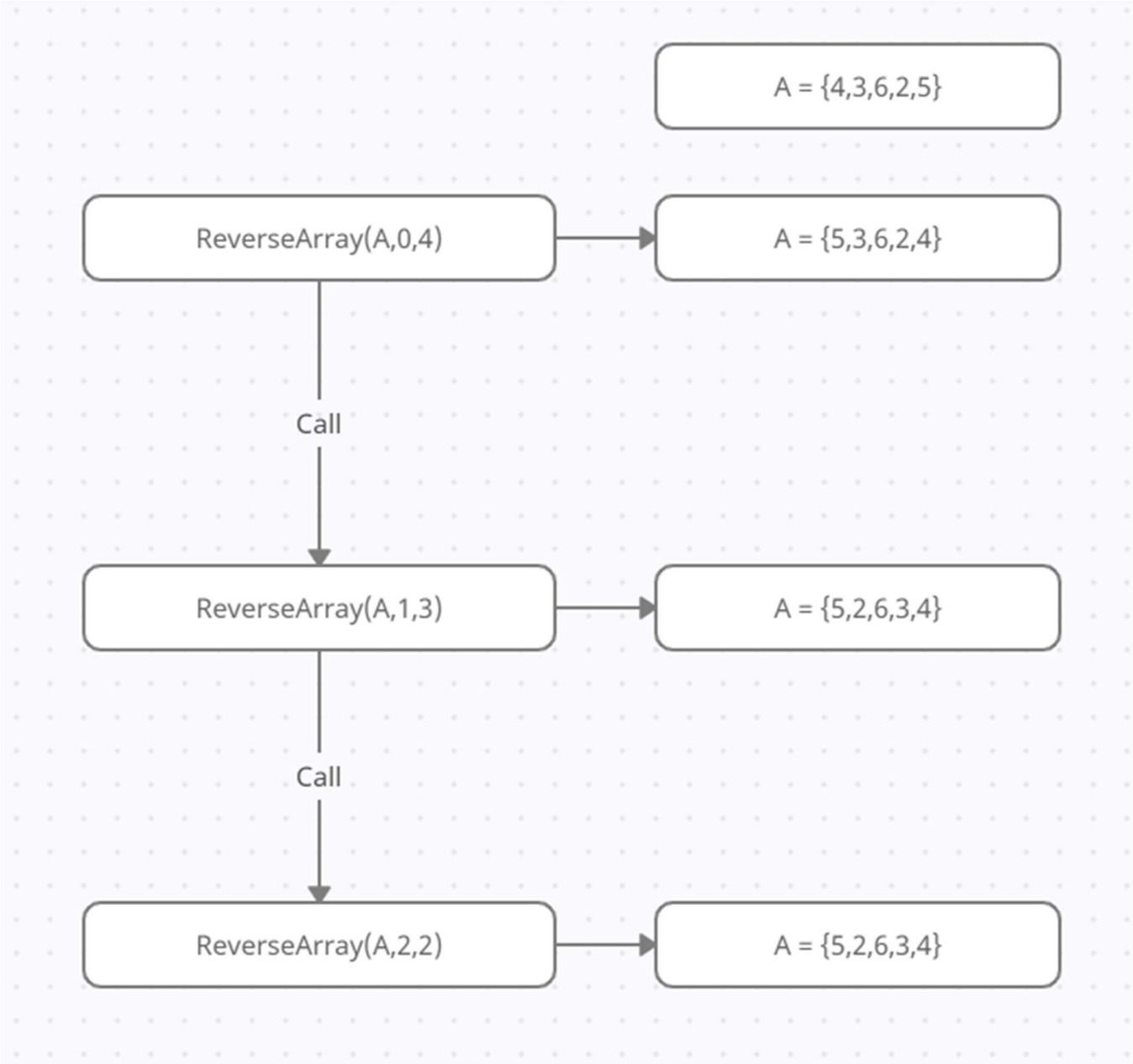
```
int max(int A[], int n)
{
    if (n <= 0 || A[n] > max(A, n - 1))
    {
        return A[n];
    }
    else
    {
        return max(A, n - 1);
    }
}
```

Description for the code:

- Get the integer array A[] and the integer n.
- First, check n is the first element or not. Then, check the A[n] is bigger than max(A, n - 1) or not:
    - If meet one of those two requirements, return A[n].
- For another situation, return recursion for max(A, n - 1)

The time complexity and space complexity are both O(n).

R-3.12



C-3.5

```
const int n{1000};
bool p[n][n]{false};

int meet(int i, int j)
{
    bool isWinner[2]{true};

    p[i][j] = true;
    p[j][i] = true;

    for (int k = 0; k < n; ++k)
    {
        if (p[i][k] == false)
        {
            isWinner[0] = false;
        }

        if (p[j][k] == false)
        {
            isWinner[1] = false;
        }

        if (isWinner[0] == false && isWinner[1] == false)
        {
            return -1;
        }
    }

    if (isWinner[0] && isWinner[1])
    {
        return -2;
    }
    else if (isWinner[0])
    {
        return i;
    }
    else if (isWinner[1])
    {
        return j;
    }

    return -1;
}

int main()
{
```

```
    int result{-1};
    int firstRand;
    int secondRand;

    for (int k = 0; k < n; ++k)
    {
        p[k][k] = true;
    }

    srand(time(NULL));
    while(result == -1)
    {
        firstRand = rand() % n;
        secondRand = rand() % n;
        if (firstRand != secondRand)
        {
            result = meet(firstRand, secondRand);
        }
    }

    if (result == -2)
    {
        cout << "Tie!" << endl;
    }
    else
    {
        cout << "Player number " << result + 1 << " won!" << endl;
    }

    return 0;
}
```

Description for the code:

- Create a 2D bool array p[n][n] and initialize them all to false.
- Change all the p[i][j] where i=j to true since they won't meet themselves.
- Start competition, each time people meet go to the meet(i, j):
    - Change p[i][j] and p[j][i] to true.
    - If row i and row j have all true, output -2 to tell main function there was a tie.
    - If row i or row j have all true, output the result as number i or number j.
    - If nobody won, output -1 to tell main function keep the loop.

C-3.10

Here is the code for SLL swap:

```
void swapSLL(string x, string y)
{
    Node *cur = head;
    Node *xprev, *xcur, *xnext;
    Node *yprev, *ycur, *ynext;
    while (cur)
    {
        if (cur->next->str == x)
        {
            xprev = cur;
            xcur = cur->next;
            xnext = cur->next->next;
        }
        if (cur->next->str == y)
        {
            yprev = cur;
            ycur = cur->next;
            ynext = cur->next->next;
        }

        cur = cur->next;
    }

    xprev->next = ycur;
    xcur->next = ynext;
    yprev->next = xcur;
    ycur->next = xnext;
}
```

Description for the code: Assume there are node X and node Y we need to swap

- Change X's previous node's next to Y.
- Change X's next to Y's next node.
- Change Y's previous node's next to X.
- Change Y's next to X's next node.

Here is the code for DLL swap:

```
void swapDLL(string x, string y)
{
    Node *cur = head;
```

```
    Node *xprev, *xcur, *xnext;
    Node *yprev, *ycur, *ynext;
    while (cur)
    {
        if (cur->next->str == x)
        {
            xprev = cur;
            xcur = cur->next;
            xnext = cur->next->next;
        }
        if (cur->next->str == y)
        {
            yprev = cur;
            ycur = cur->next;
            ynext = cur->next->next;
        }

        cur = cur->next;
    }

    xprev->next = ycur;
    xnext->prev = ycur;
    yprev->next = xcur;
    ynext->prev = xcur;
    xcur->prev = yprev;
    xcur->next = ynext;
    ycur->prev = xprev;
    ycur->next = xnext;
}
```

Description for the code: Assume there are node X and node Y we need to swap

- Change X's previous node's next to Y.
- Change X's next node's previous to Y.
- Change Y's previous node's next to X.
- Change Y's next node's previous to X.
- Change X's previous to Y's previous node.
- Change X's next to Y's next node.
- Change Y's previous to X's previous node.
- Change Y's next to X's next node.

Four steps for SLL to process and eight steps for DLL to process. As a result, the second algorithm will take more time.

Chapter 4

R-4.7

$$8nlogn \geq 2n^2$$

$$\Rightarrow \quad 8logn \geq 2n$$

$$\Rightarrow \quad logn \geq \frac{1}{4}n$$

$\Rightarrow$ When n = 16, $logn = 4$ and $\frac{n}{4} = 4$, so when n < 16, right side will be more effective.

R-4.13

$$2^{10} < 2^{logn} < nlogn < 4n < 4nlogn + 2n < 3n + 100logn < n^2 + 10n < n^3$$

R-4.18

$$O(n^2)$$

R-4.20

$$O(n^3)$$

C-4.6

```
void sort(int A[], int n, int k)
{
    int B[10000];
    int begin{0};
    int end{n - 1};
    int temp;
    for (int i = 0; i < n; ++i)
    {
        if (A[i] <= k)
        {
            temp = A[begin];
            A[begin] = A[i];
            A[i] = temp;
            begin++;
```

```
        }
        else
        {
            temp = A[end];
            A[end] = A[i];
            A[i] = temp;
            end--;
        }
    }
}
```

The running time will be O(n).