FINAL EXAM A

SECOND SEMESTER OF ACADEMIC YEAR 2023 – 2024

SOLUTION & SCORING CRITERION

1. Inheritance& File I/O (18pts)

scoring criterion: each question 2 pts

```
/* (1)请在此处实现 CubiodCake 的成员函数 cakeprice()，返回特定的某个长方体
* 形状蛋糕的价格，计算公式：蛋糕价格=体积*密度*单价。*/
double CubiodCake::cakeprice() const {                    // 2pts
    return length * width * height *get_density()* get_unitPrice();
}

/* (2)请在此处实现 CylinderCake 的成员函数 cakeprice()，返回特定的某个圆柱体
* 形状蛋糕的价格，计算公式：蛋糕价格=体积*密度*单价。*/
double CylinderCake::cakeprice()const{                    // 2pts
  return  3.14 * radius * radius * height*get_density()*get_unitPrice();
}

int main() {
    ifstream inputFile("cakes.txt");
// (3) 请在此处建立 ofstream 类对象 outputFile，并同时打开磁盘文件 totalcost.txt
    ofstream outputFile("totalcost.txt");                 // 2pts
// (4) 请修改下面 if 语句的条件，判断是否成功打开了文件
  if (!inputFile.is_open() || !outputFile.is_open()) {    // 2pts
      cerr << "Error opening files!" << endl;
```

```
    return 1;
}
double totalCost = 0;
char type;
Cake *pcake;
/* (5) 请修改下面 while 语句中的 true 条件部分，读取 cakes.txt 中每一行数据的蛋糕类型 type， 提示：
    请用操作符>>读取，该操作若读取成功返回 ture,若遇文件结束则返回 false。*/
while (inputFile >> type) {                                    // 2pts
    if (type == 'U') {
        double length = 0, width = 0, height = 0, density=0, price=0;
            // (6) 请从磁盘文件依次读入 length，width，height，price
        inputFile >> length >> width >> height>> density >> price;      // 2pts
        CubiodCake cubiodCake(length, width, height, density, price);
        double cost=cubiodCake.cakeprice();
        //cout << "the cost:" <<cost << endl;
        totalCost += cost;
    } else if (type == 'Y') {
        double radius = 0, height = 0,density=0,price=0;
            //(7) 请从磁盘文件依次读入 height，radius，price
        inputFile >> height >> radius >> density >> price;              // 2pts
        CylinderCake cylinderCake(radius,height,density,price);
        double cost=cylinderCake.cakeprice();
        //cout << "the cost:" <<cost << endl;
        totalCost += cost;
    }
    }
// (8) 请在此处将计算得到的 totalCost 的值写入磁盘文件 totalcost.txt
    outputFile << totalCost;                                    // 2pts
//（9）请关闭打开的文件
    inputFile.close();                                          // 2pts
```

```
    outputFile.close();
  return 0;
}
```

2. LinkedLists (17pts)

```
Node* subList(Node* list, int x, int n)
{
   Node* head = nullptr;              // 1pt
   Node* tail = nullptr;              // 1pt

   while (list != nullptr)            // 2pts
   {
      if (list->data >= x)            // 1pt
      {
         if (n > 0)                   // 1pt
         {
            if (head == nullptr)      // 1pt
            {
               head = tail = new Node{list->data, nullptr}; // 2pts
            }
            Else
            {
               tail->next = new Node{list->data, nullptr}; // 2pts
               tail = tail->next;     // 1pt
            }
            n--;                      // 1pt
         }
         else
         {
            break;                    // 1pt
         }
      }
      list = list->next;              // 2pts
   }

   return head;                       // 1pt
}
```

3. String or sorting (15pts)

```
void mergeSort(vector<Student>& students, int left, int right)
{                                                                  // 1pts
   if (left < right) {
```

```
        int middle = left + (right - left) / 2;                    // 2pts
        mergeSort(students, left, middle);                         // 2pts
        mergeSort(students, middle + 1, right);                    // 1pts
        merge(students, left, middle, right);
    }
}

void merge(vector<Student>& students, int left, int middle, int
right) {                                                           // 2pts
    int n1 = middle - left + 1;
    int n2 = right - middle;

    vector<Student> L(n1);
    vector<Student> R(n2);
                                                                   // 1pts
    for (int i = 0; i < n1; ++i)
        L[i] = students[left + i];
    for (int i = 0; i < n2; ++i)
        R[i] = students[middle + 1 + i];
                                                                   // 2pt
    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i].score <= R[j].score) {
            students[k] = L[i];
            ++i;                                                   // 2pts
} else {
            students[k] = R[j];
            ++j;
        }
        ++k;
    }
                                                                   // 1pt
    while (i < n1) {
        students[k] = L[i];
        ++i;
        ++k;
    }
                                                                   // 1pt
    while (j < n2) {
        students[k] = R[j];
        ++j;
        ++k;
    }
}
```

4. Big-O (18pts)

a) O(N*N) // 4.5pts

b) O(1) // 4.5pts

c) O(N) // 4.5pts

d) O(logN) // 4.5pts

5. ADT-1 & Recursion (17pts)

```
void generatePermutations(Vector<int>& nums, int index, Map<int, Vector<int>>&
permutations, int& permIndex) {
   if (index == nums.size() - 1) {                    //  2pts
      permutations[permIndex++] = nums;               //  2pts
      return;                                         //  1pt
   }

   for (int i = index; i < nums.size(); ++i) {        //  2pts
      swap(nums[index], nums[i]);                     //  1pt
      generatePermutations(nums, index + 1, permutations, permIndex);  //  3pts
      swap(nums[index], nums[i]);                     //  1pt
   }
}

void printPermutations(const Map<int, Vector<int>>& permutations) {
   for (const auto& entry : permutations) {           //  2pts
      cout << entry.first << ": ";
      for (int num : entry.second) {                  //  2pts
         cout << num << " ";
      }
      cout << endl;                                   //  1pt
   }
}
```

6. ADT-2 & Recursion (15pts)

```
bool possible(Vector<int> a, Vector<int> b, int k)
{
   if (a == b)                          //  2pts
   {
      return true;                      //  1pt
   }
   else                                 //  1pt
   {
      if (k > 0)                        //  2pts
      {
         for (int i = 0; i < a.size() - 1; i++)     //  1pt
         {
            for (int j = i + 1; j < a.size(); j++) //  1pt
            {
               swap(a[i], a[j]);                  //  1pt
               if (possible(a, b, k - 1))         //  2pts
               {
                  return true;                    //  2pts
               }
               swap(a[i], a[j]);                  //  1pt
            }
         }
      }
      return false;                                //  1pt
   }
}
```