上海财经大学《程序设计基础》上机考试卷(A)

参考答案及评分标准

（ 2019 至 2020 学年 第 1 学期 ）

1. (循环控制。 15 分）

根据输入正整数 num（0<num<10），要求通过循环控制输出数字塔（上下对称）。

例如，num=4 时，输出图案如下。

```
   1
  121
 12321
1234321
 12321
  121
   1
```

```
// 答案如下：
#include <iostream>
using namespace std;

int main(){
   int num;
   cout << "Please input the number (0<num<10): ";
   cin>>num;
   if(num >0 && num<10)                          //    2分
   {
      for(int i = 1; i<= num; i++){              //    2分
         for(int j = 0; j<num - i; j++) cout<<' ';   //    1分
         for(int j = 1; j<i; j++) cout<<j;       //    1分
         for(int j = i; j>0; j--) cout<<j;       //    1分
         cout<<endl;                             //    1分
       }
      for(int i = num -1; i>0; i--){             //    2分
         for(int j = 0; j<num - i; j++) cout<<' ';   //    1分
         for(int j = 1; j<i; j++) cout<<j;       //    1分
         for(int j = i; j>0; j--) cout<<j;       //    1分
         cout<<endl;                             //    1分
       }
   }
```

```
    else                                                    //   1分
        cout<<"input data error!"<<endl;
    system("pause");
    return 0;
}
```

2. (函数调用。 15 分）

编写名为 calc 函数，函数实现如下功能：计算数组元素的平均值，并且将此数组中的偶数数据进行求和运算，奇数数据进行求积运算，不得采用全局变量。

```
// 答案如下:
#include <iostream>
using namespace std;
void calc(int *a,int *x,int *y,float &vavg)     // 4 分
{
    *x=0;                                       // 1 分
    *y=1;                                       // 1 分
    vavg=0.0;                                   // 1 分
    for(int i=0; i<10; i++)                     // 2 分
    {
        if(a[i]%2)                              // 2 分
            *y=*y*a[i];                         // 1 分
        else
            *x=*x+ a[i];                        // 1 分
        vavg=vavg+ a[i];                        // 1 分
    }
    vavg=vavg/10;                               // 1 分
}

int main()
{
    int   x,y,a[10]= {1,2,3,5,6,7,8,8,9,10};
    float   vavg;
    calc(a,&x,&y,vavg);
    cout<<"平均值="<<vavg<<"偶数和="<<x<<"奇数积="<<y<<endl;
    system("pause");
    return 0;
}
```

3. (改错题，数组与字符串，15 分) 纠正下列排序算法中的错误（请勿重写该排序算法）。

```
//答案如下：
#include <iostream>
using namespace std;

void swapn(int &a, int &b)        // 使用引用，5 分
{
    int tmp = a;
    a = b;
    b = tmp;
}

int main()
{
    int a[] = {5, 1, 7, 9, 2, 4, 10, 16};
    int nsize = sizeof(a) / sizeof(int);

    for(int i = 1; i <= nsize; i++){
        bool flag = false;
        for(int j = 0; j < nsize - i; j++){
            if(a[j] > a[j+1]){
                swapn(a[j], a[j+1]);
                flag = true;
            }
        }
        if(!flag)       break;                  // 取反，5 分
    }

    for(int i = 0; i < nsize; i++){             // 正确访问数组元素，5 分
        cout << a[i] << " ";
    }
    cout << endl;
    system("pause");
    return 0;
}
```

4. (改错题，指针与动态数组，15 分)

```
// 答案如下：
dynamicArray *applyMemory(int n)
{
    dynamicArray *pa;
    pa=new dynamicArray;
    pa->capacity=n;
    pa->num=0;
    pa->arr=new int [n];          // 为数组申请动态内存    5 分
    return pa;
}

// 函数: insertElementAtEnd
// 作用：insertElementAtEnd(dynamicArray *& pa,int elem)
//       在 pa 指向的动态数组的末尾添加一个新元素 elem。
//       注意：增添新元素时，可能会超出原来容量限制，
//       此时要求申请一个新的空间，其中的数组应扩充为原数组空间的两倍。
//       同时，要将数组内容复制到新的空间，并更新 capacity 的值。
// 用法示例：insertElementAtEnd(p,200);    即增添一个元素 200 到 p 所指向的动态数组中。

void insertElementAtEnd(dynamicArray *& pa,int elem){
    dynamicArray *tempp;
    if(pa->num < pa->capacity){
        pa->arr[pa->num]=elem;
        pa->num++;
    }
    else{
        tempp=new dynamicArray;
        tempp->arr=new int[2*(pa->capacity)];
        tempp->capacity = 2*(pa->capacity);   //应给 capacity 赋值   5 分
        for(int i=0;i<pa->num;i++)
            tempp->arr[i]=pa->arr[i];
        tempp->arr[pa->num]=elem;
        tempp->num++;                         //修改为  tempp->num = pa->num + 1;    5 分
        delete [] pa->arr;
        delete pa;
        pa = tempp;
    }
}
```

5.（递归与搜索。 **15** 分）

快速排序是一种非常高效的排序算法，其充分利用了分治法和递归两种算法思想，其衍生算法也可以用于解决其它非排序问题。例如，给定一个数组，假设其中的数据是随机的，如何快速的找到数组中第 **k** 小的数？我们可以通过对快速排序进行改造以解决该问题，其核心思路为：每次以数组序列中第一个元素作为中间数 **mid**，将序列中小于等于 **mid** 的值放在 **mid** 左边，而大于它的值放在 **mid** 右边。不妨设小于等于 **mid** 的个数为 **m**，若 **k < m**，则对左边序列递归做相同操作；若 **k > m**，则对右边序列递归寻找序列中第 **k-m** 小的元素；若 **k = m**，则 **mid** 即为第 **k** 小的元素。

例如，数组 **a** 的下标和对应元素如下表：

下标	**0**	**1**	**2**	**3**	**4**	**5**	**6**
对应元素	**3**	**1**	**2**	**6**	**8**	**7**	**9**

若我们以 **6** 作为 **mid** 值，通过 **divide** 函数，我们可以让它左边的元素都比它小，而右边的元素都比它大。如果我们需要找到整个数组中第 **2** 小的元素，则此时该元素肯定在 **6** 的左边，我们只需递归地寻找在下标为 **0-2** 范围内第 **2** 小的元素即可；如果我们需要找到整个数组中第 **5** 小的元素，则由于小于等于 **6** 的元素数量只有 **4** 个，第 **5** 小的元素必然在 **6** 右边，此时，只需要递归地在下标范围为 **4-6** 的范围内寻找第 **1** 小（即最小）的元素，就是全局第 **5** 小的元素。试用递归的形式描述该过程，并实现该算法。

// 答案如下：

```
#include <iostream>
using namespace std;

int search_k(int a[], int size, int k);                    // 负责寻找数组 a 中第 k 小元素函数
int quick_search(int a[], int low, int high, int k);       // 基于快速搜索思想的递归函数
int divide(int a[], int low, int high);                    // 分段函数，将数组分割为小于等于 a[low]和大于
                                                           //a[low]两段，a[low]在中间的函数,返回 a[low]所在的位置。

int main()
{
    int numbers[] = {5, 8, 3, 1, 2, 7, 8, 10,15, 13};     // numbers 为待搜索的数组
    int nsize = sizeof(numbers) / sizeof(int);             // nsize 为数组中元素的个数

    cout << search_k(numbers, nsize, 3) << endl;           // 输出第 3 小的元素，应该为 3;
    cout << search_k(numbers, nsize, 5) << endl;           // 输出第 5 小的元素，应该为 7;
    cout << search_k(numbers, nsize, 7) << endl;           // 输出第 7 小的元素，应该为 8.
    system("pause");
    return 0;
}

int search_k(int a[], int nsize, int k)
{
    return quick_search(a, 0, nsize-1, k);
}
```

```
int quick_search(int a[], int low, int high, int k)
{
    if(low >= high) return a[low];          // 1 分
    int mid = divide(a, low, high);         // 2 分
    int m = mid - low + 1;                  // 2 分

    if(m == k) return a[mid];               //1 分
    else if(m > k) return quick_search(a, low, mid-1, k);      // 2 分
    else return quick_search(a, mid+1, high-1, k-m);           // 2 分
}

int divide(int a[], int low, int high)
{
    // 将该函数补充完整，共 5 分。教材第 141 页，分段函数实现，抄写即可。
    int k = a[low];
    do{
        while(low < high && a[high] >= k) --high;      // 1 分
        if(low < high) {a[low] = a[high]; ++low;}      // 1 分
        while(low <high && a[low] <= k) ++low;         // 1 分
        if(low < high) {a[high] = a[low]; --high;}     // 1 分
    }while(low != high);
    a[low] = k;                                        // 1 分
    return low;
}
```

6.（类。 13 分）

集合是数学中一个基本概念，为若干元素构成的整体。本题要求建立一个元素为整数的集合类 Set，该类提供以下有关集合的基本操作：

1) Set::Set(); // 构造函数，建立一个空集。
2) bool Set::ismember(int x); //判断并返回 x 是否在当前集合。
3) void Set::insert(int x); //把 x 加到当前集合。
4) Set Set::unite(const Set &); // a.unite(b) 返回 a 跟 b 的并集。
5) Set Set::intersect(const Set &); // a.intersect(b) 返回 a 跟 b 的交集。

我们假设元素的范围是 0 到 99 之间的整数，于是可以用一个布尔数组记录每一个元素是否在集合里面。请补充完成以下缺少的代码。

//答案如下：

```
Set::Set()                              // 3 分
{
    for (int i = 0; i < R; ++i)
```

```
        table[i] = false;
}
bool Set::ismember(int x) const          // 2 分
{
    return table[x];
}
void Set::insert(int x)                  // 2 分
{
    table[x] = true;
}
Set Set::unite(const Set &s)             // 3 分
{
    Set ret;
    for (int i = 0; i < R; ++i)
        if(table[i] || s.table[i])
            ret.table[i] = true;
    return ret;
}
Set Set::intersect(const Set &s)         // 3 分
{
    Set ret;
    for (int i = 0; i < R; ++i)
        if (table[i] && s.table[i])
            ret.table[i] = true;
    return ret;
}
```


7. （链表。12 分）

本题是一个简化的文本处理程序，其中使用链表作为数据结构来存储单词信息。原始文本信息存放在 TestData.txt 文件，处理程序从文件中逐行读取字符串，提取单词并建立每一行的单词链表。注意：本题中也使用行链表来保存文本行的信息。

单词、行和文本的数据结构定义如下：

```
struct Word{            // 单词结点的结构体
    char *str;          // 单词字符串指针
    Word *next;         // 指向下个单词结点的指针
};
struct Line{            // 文本行的结构体
    int lineNo;         // 行号
    int wordNum;        // 行包含的单词数量
    Word *head;         // 指向本行的第 1 个单词和最后一个单词的结点指针
    Word *rear;         // 指向本行的最后一个单词的结点指针
    Line *next;         // 指向下个行结点的指针
```

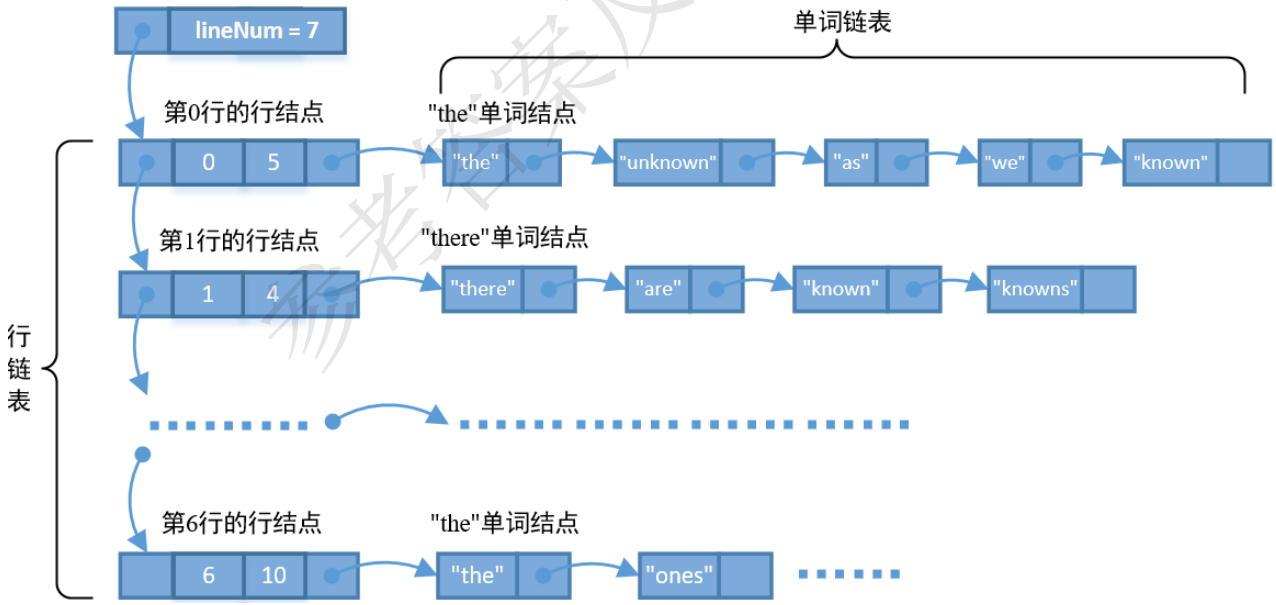
```
};
struct Text{              // 文本的结构体
    int lineNum;          // 文本行数
    Line *head;           // 指向第 1 个（即第 0 行）行结点的指针
    Line *rear;           // 指向最后一行的行结点指针
};
```

程序完成如下功能：

1）通过 **createLineList** 和 **createWordList** 函数建立行链表和单词链表：首先从文本文件中每次读入一行字符串，然后生成行结点，扫描并逐个提取其中的单词，生成相应的单词结点，再插入到本行对应的单词链表的尾部。假设文本文件内容如下：

"**The** unknown as we know,
there are known knowns.
There are things we know we know.
We also know there are known unknowns.
That is to say we know there are some things we do not know.
But there are also unknown unknowns,
the ones we do not know we do not know."

当程序开始运行时，行链表和单词链表都是空的。程序调用 **processLineString** 函数从第 **0** 行开始扫描，首先生成第 **0** 行的行结点，并通过 **processWord** 提取“**the**”这个单词字符串，然后生成 **the** 单词结点，并插入到单词链表中，循环处理直到第 **0** 行结束。单词表的结构信息示意图如下所示。



行链表和单词链表示意图

2）遍历第 1）步生成的行链表和单词链表，输出每行单词的信息：包括行号、每行单词数量和行中每个单词的字符串。

3）在程序结束之前，应用循环和 **delete** 清理程序中所有动态申请的存储空间。

注意：本题目录下有 3 个文件，7.cpp, 7_demo.exe 和 TestData.txt。7.cpp 是源程序，TestData.txt 是要扫描的文本文档，务必把它和 7.cpp 放在同一目录下，而 7_demo.exe 是可以执行的演示程序，可以鼠标双击它看运行结果。

题目要求补充实现 7.cpp 的 3 个函数 createTextData, processLineString 和 printTextData 中缺少的代码。程序 7.cpp 的代码如下：

```
// 答案如下：
#include <iostream>
#include <cstring>
#include <fstream>
using namespace std;

struct Word{              // 单词结点的结构体
    char *str;           // 单词字符串指针
    Word *next;          // 指向下个单词结点的指针
};
struct Line{              // 文本行的结构体
    int lineNo;          // 行号
    int wordNum;         // 行包含的单词数量
    Word *head;          // 指向本行的第 1 个单词和最后一个单词的结点指针
    Word *rear;          // 指向本行的最后一个单词的结点指针
    Line *next;          // 指向下个行结点的指针
};
struct TextData{       // 文本的结构体
    int lineNum;       // 文本行数
    Line *head;        // 指向第 1 个（即第 0 行）行结点的指针
    Line *rear;        // 指向最后一行的行结点指针
};

bool createTextData(TextData &td,const char fileName[]); // 从文件读取字符串，生成结构化文本数据
void processLineString(Line *nl,const char lineString[]); // 提取一行字符串中的单词，插入单词表
void processWord(Word *wd,char *str);// 处理提取的单词,生成单词结点，并插入到单词链表尾部。
void printTextData(const TextData &td); // 输出文本数据信息
void cleanTextData(TextData &td);        // 释放行链表和单词链表
bool isPunctuation(char s); // 检查字符 s 是标点符号或是分隔符吗？

int main()
{
    TextData td ={0,NULL,NULL};
```

```
    const char *fileName = "TestData.txt";
    createTextData(td,fileName);
    cout<<endl<<"------------文件处理完成!--------------"<<endl<<endl;
    printTextData(td);
    cleanTextData(td);
    cout<<endl<<"----------  数据清理完成!-------------"<<endl<<endl;
    system("pause");
    return 0;
}

bool createTextData(TextData &td,const char fileName[])
{   // 从文件逐行读取字符串
    char lineString[256];
    ifstream textFile;

    textFile.open(fileName); // 打开文件
    if (textFile.fail()){       // 判断文件打开是否失败？
        cout<<"文件打开失败！检查当前目录下是否有 TestData.txt 文件！";
        return false;
    }
    else{ // 打开文件成功，开始循环：每次从文件中读取一行到字符串 lineString 中
        while(textFile.getline(lineString,256)) {
            cout<<lineString<<endl;
            Line *nl = new Line;
            nl-> lineNo = nl->wordNum = 0;      // 1 分
            nl->next = NULL;                    // 1 分
            nl->head = nl->rear = NULL;          // 1 分
            if(td.rear){                         // 2 分
                td.rear->next = nl;
                td.rear = nl;
            }
            else                                // 1 分
                td.head = td.rear = nl;
            // 提取字符串 lineString 中的所有单词，
            // 生成相应的单词结点，并插入到该行单词链表的尾部。
            processLineString(nl,lineString);
            nl->lineNo = td.lineNum++;
        }
        textFile.close();   // 处理完成后，关闭文件
        return true;
    }
}

void processLineString(Line *nl,const char lineString[])
```

```
{    // 提取一行字符串中的单词，插入单词链表
     char str[256];
     int i = 0, j = 0;

     while(lineString[i]){   // 扫描一行字符串直到'\0'为止
          while(isPunctuation(lineString[i])) i++;   // 跳过标点符号和分割符
          while(!isPunctuation(lineString[i])&&lineString[i])
               str[j++] = lineString[i++];
          if (j>0){   // 提取到一个单词，使用 str 保存它的字符串
               str[j] = '\0';
               j = 0;

               Word *wd = new Word;
               if(nl->rear){                    // 2 分
                    nl->rear->next = wd;
                    nl->rear = wd;
               }
               else                             // 1 分
                    nl->head = nl->rear = wd;

               processWord(wd,strlwr(str));// 处理一个提取的单词
               nl->wordNum++;
          }
     }
}

// 处理提取的单词：生成相应的单词结点，并插入到单词链表尾部
void processWord(Word *wd,char *str)
{
     // 提取的单词不在单词表中，生成单词结点并插入到单词链表
     wd->str = new char[strlen(str)+1];
     strcpy(wd->str,str);
     wd->next = NULL;
}

void printTextData(const TextData &td)
{
     Line *nl = td.head;

     while(nl){                              //       1 分
          cout<<" 第"<<nl->lineNo<<"行，共有单词："<<nl->wordNum<<"个！"<<endl;
          Word *wd = nl->head;
          while(wd){                          //       1 分
               cout<<"      单词  "<<wd->str<<endl;
```

```
            wd = wd->next;
        }
        nl = nl->next;                          //      1 分
    }
}

void cleanTextData(TextData &td)
{
    Line *nl = td.head;

    while(nl)
    {
        td.head = nl->next;
        Word *wd = nl->head;
        while(wd)
        {
            nl->head = wd->next;
            delete wd->str;
            delete wd;
            wd = nl->head;
        }
        delete nl;
        nl = td.head;
    }
}

bool isPunctuation(char s) //  检查字符 s 是标点符号或是分隔符吗？
{
    char punctuationSet[] = {' ','\t',';',',','.','!',':','\?','\"','\'','-','\0'};
    int i=0;

    while(punctuationSet[i])
        if (s == punctuationSet[i++])
            return true;

    return false;
}
```