

# 上海财经大学《程序设计基础》上机考试卷(A)

( 2019 至 2020 学年 第 1 学期 )

学号

姓名

机器号

答题及答案提交要求:

- (1) 务必在 D 盘上创建一个**答题文件夹 (目录)**, 该文件夹的名字用你的学号+姓名命名, 如你的学号为 2016000123, 姓名为“李明”, 则你需要创建一个目录“2016000123 李明”。**将你做的答案保存在 D 盘你的文件夹下。**
- (2) 请将你修改完成后的 **1.cpp, 2.cpp, 3.cpp, 4.cpp, 5.cpp, 6.cpp, 7.cpp** 保存在 D 盘你的文件夹下。

注意事项: 以下试题中没留出足够空行, 应根据需要确定代码长度。

## 1. (循环控制。 15 分)

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

例如,  $\text{num}=4$  时, 输出图案如下。

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

程序如下:

```
#include <iostream>
using namespace std;
int main()
{
    int num;
    cout << "Please input the number (0<num<10): ";
    cin >> num;
    //请补充完成以下缺少的代码 15 分

    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。

//主函数如下所示，不得更改。 编写 `calc` 函数，将下面程序补充完整。

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

int main()
{
    int x = 0, y = 0, a[10] = {1,2,3,5,6,7,8,8,9,10};
    float vavg = 0;
    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）。

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

void swapn(int a, int b)
{
    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;
    }

    for(int i = 1; i <= nsize; i++){
        cout << a[i] << " ";
    }
    cout << endl;
    system("pause");
    return 0;
}

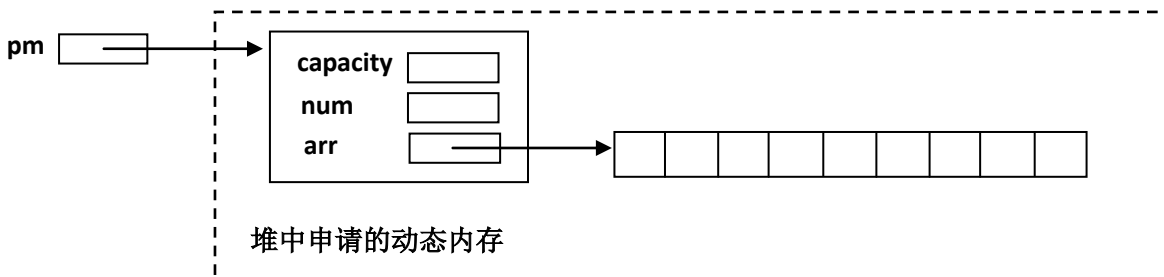
```

#### 4. (改错题，指针与动态数组，15 分)

以下程序使用动态内存来存储一组整型数（数据结构如图所示）。该程序首先建立一个动态数组，之后往该数组中增添若干新的元素，并显示数组的内容。程序结束之前，释放动态数组所占用的内存。请分别修改函数 `applyMemory` 以及函数 `insertElementAtEnd` 中存在的错误。

本题的数据结构图示：

`dynamicArray *pm;` //此处 `pm` 是指向结构体类型(`dynamicArray`)变量的指针。



```

#include <iostream>
using namespace std;

```

```

struct dynamicArray {
    int capacity;    //记录数组最大可存储的元素个数
    int num;        //记录数组当前的元素个数
    int *arr;       //指向数组的指针
};

```

```
dynamicArray *applyMemory(int );
void insertElementAtEnd(dynamicArray *& pa,int elem);
void displayArray(const dynamicArray *);
void deleteMemory(dynamicArray *);
```

```
int main()
{
    int n=10;
    dynamicArray *pa = applyMemory(n);
    // 为数组赋值
    for(int i=0;i<n;i++){
        pa->arr[i]=i+1;
        pa->num++;
    }
    cout << "The Array is:\n";
    displayArray(pa);
    //在数组中增加 50 个元素
    for(int i=0;i<50;i++)
        insertElementAtEnd(pa,i+1);

    cout << "\nThe Array after adding 100 elements is:\n";
    displayArray(pa);
    deleteMemory(pa); //释放数组占据的内存
    system("pause");
    return 0;
}
```

/\* 函数： applyMemory

\* 作用： applyMemory(int num)为元素个数是 num 的数组在堆中申请内存空间，  
\* 并记录空间容量的大小，初始化数组元素个数为 0。

\* 用法： p = applyMemory(10);

\*/

```
dynamicArray *applyMemory(int n)
```

```
{
    // 请改正以下程序中存在的错误，或补充缺少的代码（5 分）
    dynamicArray *pa;
    pa=new dynamicArray;
    pa->capacity=n;
    pa->num=0;

    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: 此处形式参数 pa 是结构体类型 (dynamicArray) 指针的引用。

\* 注意: 增添新元素时, 可能有两种情况:

\* (1) 原来的存储容量大于当前元素总数, 此时直接添加新的元素, 不需要扩充数组的内存空间。

\* (2) 原来的存储容量已经用满了, 增添新元素会超出原有的容量限制, 此时需要申请一个新的更大空间, 本题要求此时数组空间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, 此处 p 是指向

结构体 (dynamicArray) 变量的指针。

\*/

```
void insertElementAtEnd(dynamicArray *& pa,int elem){
```

```
    dynamicArray *tempp;
```

```
    if(pa->num < pa->capacity){ //如果数组当前元素个数小于数组当前最大容量
```

```
        pa->arr[pa->num]=elem;
```

```
        pa->num++;
```

```
    }
```

```
    else{
```

```
        // 请改正以下存在的两处错误, 一处是忘记了给结构体成员赋值, 一处是赋值错误, (10 分)
```

```
        tempp=new dynamicArray;
```

```
        tempp->arr=new int[2*(pa->capacity)];
```

```
        for(int i=0;i<pa->num;i++)
```

```
            tempp->arr[i]=pa->arr[i];
```

```
        tempp->arr[pa->num]=elem;
```

```
        tempp->num++;
```

```
        delete [] pa->arr; // 释放原数组占据的内存空间(1)
```

```
        delete pa; // 释放原数组结构体占据的内存空间(2)
```

```
        pa = tempp; //修改 pa
```

```
    }
```

```
}
```

/\* 函数: displayArray

\* 作用: 依次显示数组中的元素。

\* 用法: displayArray(pa);

\*/

```
void displayArray(const dynamicArray *p){
```

```
    int n=p->num;
```

```
    for(int i=0;i<n-1;i++)
```

```
        cout << p->arr[i] << ",";
```

```
    cout << p->arr[n-1] << endl;
```

```
}
```

```

/* 函数: deleteMemory
* 作用: 释放数组在堆中占据的内存。
* 用法: deleteMemory(p);
*/
void deleteMemory(dynamicArray *p){
    delete [] p->arr;
    p->arr=NULL;
    delete p;
    p=NULL;
}

```

## 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。

```

#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)
{
    // 将下面代码修改并补充完整，共 10 分。
    return 0;
}

int divide(int a[], int low, int high)
{
    //将下面代码修改并补充完整，共 5 分。
    return 0;
}

```

## 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。

```

#include <iostream>
using namespace std;
#define R 100
class Set

```

```

{
private:
    bool table[R];           // 记录每一个元素是否在集合里面，数组大小为元素范围。
public:
    Set();                   // 构造函数，新建的集合应设为空集。
    bool ismember(int x) const; // 返回 x 是否在集合里面。
    void insert(int x);       // 把 x 加到当前集合。
    Set unite(const Set &);   // a.unite(b) 返回 a 跟 b 的并集。
    Set intersect(const Set &); // a.intersect(b) 返回 a 跟 b 的交集。
};

void display(const Set &s);   // 把集合输出到屏幕。
int main()
{
    Set a, b;
    for (int i = 0; i < 100; ++i)
    {
        if (i % 9 == 0)
            a.insert(i);       // a = {0, 9, 18, ..., 99} 为 100 内的 9 的倍数。
        if (i % 12 == 0)
            b.insert(i);       // b = {0, 12, 24, ..., 96} 为 100 内的 12 的倍数。
    }
    Set c = a.unite(b);
    display(c);                // {0, 9, 12, 18, 24, 27, 36, 45, 48, 54, 60, 63, 72, 81, 84, 90, 96, 99}
    Set d = a.intersect(b);
    display(d);                // {0, 36, 72}
    system("pause");
    return 0;
}

void display(const Set &s)
{
    bool isfirst = true;
    cout << "{";
    for (int i = 0; i < R; ++i)
        if (s.ismember(i))
        {
            if (!isfirst)
                cout << ", ";
            cout << i;
            isfirst = false;
        }
    cout << "}" << endl;
}

```



// 请补充完成以下函数。

```
Set::Set()                // 3 分
{
}
bool Set::ismember(int x) const    // 2 分
{
}
void Set::insert(int x)           // 2 分
{
}
Set Set::unite(const Set &s)      // 3 分
{
}
Set Set::intersect(const Set &s)  // 3 分
{
}
```

## 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;           // 指向本行的首个单词的结点指针
    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;           // 指向首行（即第 0 行）行结点的指针
    Line *rear;           // 指向最后一行的行结点指针
};
```

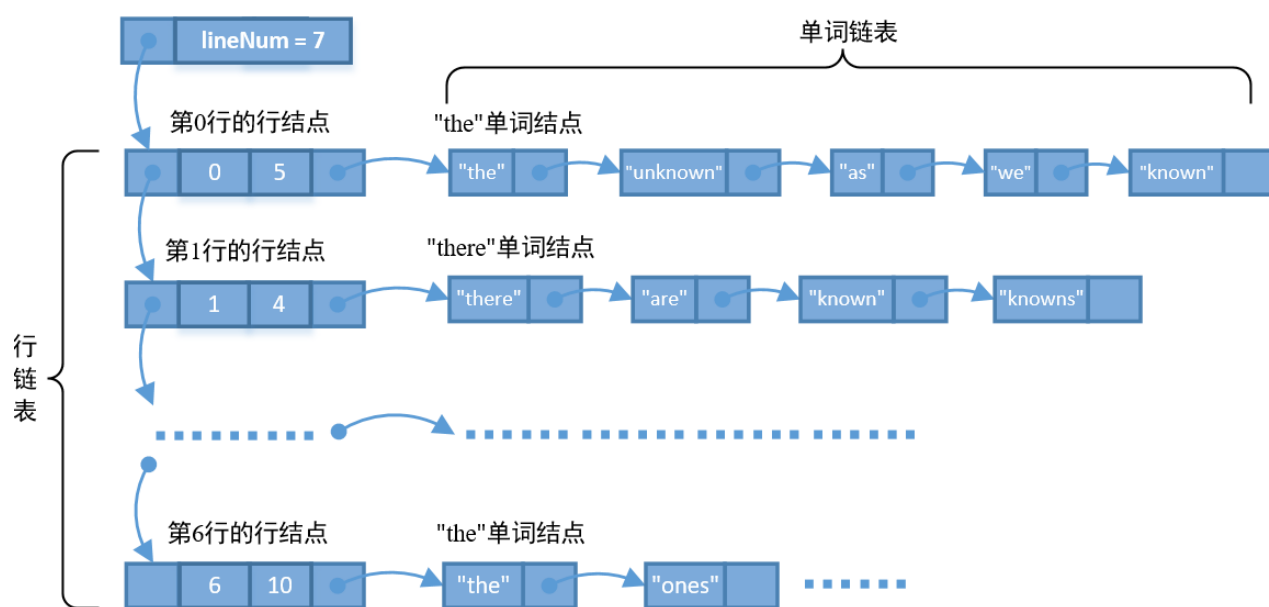
程序完成如下功能：

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。文本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` 是要扫描的文本文件，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：

```

#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，插入单词链表
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);      // 释放行所有动态申请的存储空间
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;
        // 行结点赋初值并插入到行链表的尾部
        // 请补充完成以下缺少的代码 6 分

        // 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;
            // 把生成的单词结点插入到单词链表尾部
            // 请补充完成以下缺少的代码 3 分

            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;
```

```
    // 遍历行链表和单词链表，输出行链表和单词链表信息
```

```
    // 请补充完成以下缺少的代码 3 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;
}
```