期末复习课

期末考试安排：2021-1-4 14:00-16:00

- 方式：闭卷，2小时，提供草稿纸
- 试卷题型和分值
  - 简答题（15分）

    基本概念
  - 程序分析（50分）

    分析程序错误，说明原因，不需要修改程序。

    判断程序结果（从main函数开始分析，流程图帮助分析）
  - 编程题（35分）

    难度不会超过平时的练习。

本学期主要内容

- 过程式程序设计的本质，良好的程序设计风格
- 基本数据类型、操作符（算术、关系、逻辑、?:）
- 流程控制（复合语句、选择语句、循环语句、break、continue）
- 基于过程抽象的程序设计（功能分解）、函数的调用+参数传递，递归函数，标识符的生存期和作用域
- 构造数据类型：数组、结构类型、指针、引用
- 面向对象部分了解什么是数据抽象

课程内容回顾——过程式程序设计的本质

过程式程序设计=算法+数据结构

程序设计的步骤：需求分析、系统设计、编程实现、测试与调试以及运行和维护。

需求分析+系统设计：算法和数据结构的设计，具体问题具体分析。

课程内容回顾——基本数据类型、表达式

整型、实数、字符、逻辑、void （数据类型的混合运算）

常量（字符常量、字符串常量）、符号常量、变量

操作符：算术、关系与逻辑、?:

- 实数之间的比较，因为有可能不能精确表示，所以不能直接用“==”，可以用近似比较：p<1e-10 &&p>-1e-10，替代 p==0
- / ：结果和操作数类型有关
- / 和 % 可以用于获取整数位数、数字等

最简单的算法：一个表达式可以完成功能。

课程内容回顾——程序流程

流程图——可以有效地帮助理解程序的流程。

编写程序前，可以尝试先画流程图

1、顺序流程

2、分支流程（if、 if...else...、switch/case/default）

3、循环流程（while、do-while、for）

4、流程的嵌套、复合语句的使用{...}、
流程的打断（break、continue）

5、结构化程序设计

课程内容回顾——函数

在解决复杂问题时，程序设计过程是：自顶向下，逐步精化

这一过程涉及到功能分解，将程序分解成若干子功能。

子功能对应的概念——子程序

子程序的实现——函数

函数头、功能抽象——函数的调用者需要了解的信息

函数名、形参（传递什么数据）、返回类型

函数体：封装了具体的实现

函数的嵌套调用——递归（递归条件，结束条件）

递归函数：能描述出算法，就能编写出代码。

课程内容回顾——标识符的生存期和作用域

生存期：面向对象里，对象的创建可以直观地体现。

静态：全局变量，static变量具有静态生存期。

自动：局部变量和函数的形式参数一般具有自动生存期。

动态：动态变量具有动态生存期。

作用域：

全局：具有全局作用域的标识符能在程序的任何地方访问。使用全局标识符时，若该标识符的定义点在其它源文件中或在本源文件中使用点之后，则在使用前需要声明它们。

局部：函数定义或复合语句中，从标识符的定义点开始到函数定义或复合语句结束之间的程序段。在局部变量的作用域范围内，同名全局标识符不可直接访问，可以通过::访问

课程内容回顾——函数的参数传递方式

- 值传递：形参和实参分别拥有存储空间，各有生存期；形参的改变不会影响实参；

  形参定义形式（变量定义）；实参（变量名）　void f(int x);　f(m);

- 指针传递：避免传递大量数据，形参指向了实参的空间，通过形参可以修改实参空间的数值。

  形参定义形式（指针变量定义、不带大小的数组定义+数组元素个数）；

  实参（指针变量名，数组名+数组大小）

  如果数组元素个数可以有其他方式判断，可以不用传递数组元素个数

  void f(int * x); f(&m); void f(int x[],int n); f(a,10);

  void f(char * s); f(str);

- 引用传递：形参就是实参的别名。修改形参的值，就是修改实参

  形参定义（引用定义），实参（变量名） void f(int & x); f(m);

为了避免指针、引用参数传递的副作用，形参可以用const修饰。

课程内容回顾——数组

- 一维数组是常见的形式
  - 整型数组、实数数组需要用一重循环完成输入、输出；
  - 字符数组可以整体输入、整体输出；（因为有结束符\0）
- 数组元素的访问（不要越界）
  - 下标访问最直观
  - 指针访问：动态数组
  - 注意不要越界：下标是从0开始
  - 字符数组有一个'\0', strlen(str)返回的是有效字符个数
  - 在字符拷贝中，要特别注意，字符串的结尾要有'\0'
- 二维数组用下标访问，可读性好

课程内容回顾——指针、引用

- 一级指针是常见的形式
  - 输出整型指针和实数指针指向的内容：* 指针
  - 直接输出字符指针：将指针指向的字符串输出；如果是用*操作符，是输出一个字符。
- 指针和数组：通过指针访问数组、通过指针访问动态变量（new 出来的动态变量，需要delete掉）
- 指针和结构：单链表
- 指针和函数：参数传递、返回类型
- 引用就是变量的别名，在参数传递中有副作用，用const可以防止指针和引用的副作用。

综合练习-删除重复元素（一）

- 数组存放：
  - 在同一个数组上，通过移位，删除重复元素
  - 利用另外一个数组，在拷贝的时候，跳过重复元素

数组访问及输出需要注意什么？

整型数组、实数数组、字符数组

将删除元素封装到一个函数中，函数头如何定义？

(1) 如果要求返回删除元素的个数

(2) 如果要求返回一个新的数组

综合练习-删除重复元素（二）

- 单链表存放：
  - 删除结点，参考教材。

    定义两个指针p,q，初始p指向头结点，q指向下一个结点；

    如果p,q指向结点的content相同，则p指向q的下一个结点，删除q

    如果p,q指向结点的content不同，则p,q均移动一个结点

  - 删除用函数封装，函数头如何定义

    void del(Node * head); // 什么时候需要定义成 Node * &

综合练习-删除重复元素（三）

- 特殊情况：

  输入：10->11->11->12->12->12->15

  输出：10->11->12->15

  如何处理？

  （1）如何读入数据

  第三次测验中，分数的读入 12/24 3/20 +

  ```
  int a1,b1,a2,b2; char c1,c2;
  cin>>a1>>c1>>b1>>a2>>c1>>b2>>c2;
  ```

  （2）数据用什么类型存放

OOP练习：链表实现集合类IntSet

第10次作业用单链表实现集合，现在用面向对象的方法定义一个元素类型为int、元素个数不受限制的集合类

class IntSet

{<成员描述>};

分析：

数据成员：链表的头指针、元素个数

成员函数：

1、构造函数、拷贝构造函数、析构函数

2、操作接口：集合的操作

OOP练习3：链表实现集合类IntSet

bool is_empty() const; //判断是否为空集。

int size() const; //获取元素个数。

bool is_element(int e) const; //判断e是否属于集合。

bool is_subset(const Set& s) const; //判断s是否包含于集合。

bool is_equal(const Set& s) const; //判断集合是否相等。

void display() const; //显示集合中的所有元素。

Set& insert(int e); //将e加入到集合中。

Set& remove(int e); //把e从集合中删除。

Set union2(const Set& s) const; //计算集合的并集。

Set intersection(const Set& s) const; //计算集合的交集。

Set difference(const Set& s) const; //计算集合的差。

OOP练习：链表实现集合类IntSet

```
bool IntSet::is_element(int e) const     //  判断元素是否属于集合
{     for (Node *p=head; p!=NULL; p=p->next)   // for循环遍历单链表
          if(p->value == e) return true;
      return false;
}
Set& Set::insert(int e)
{     if(!is_element(e))
      {     Node *p=new Node;
            p->value = e;          p->next = head;          head = p;
            count++;
      }
      return *this;
}
```

OOP练习：链表实现集合类IntSet（续）

```
IntSet IntSet::union(const IntSet& s) const   // 求并集
{    IntSet set(s); // 定义拷贝构造函数
     Node *p=head;
     while (p!=NULL)    // while循环遍历单链表
     {    if (!set.is_element(p->value))
               set.insert(p->value);
          p=p->next;
     }
     return set;
}
```

OOP练习：链表实现集合类IntSet（续）

```
IntSet IntSet::intersection (const IntSet& s) const // 求交集
{     IntSet set;  //空集合
      Node *p=head;
      while (p!=NULL)     // while循环遍历单链表
      {     if ( s.is_element(p->value))
                  set.insert(p->value);
            p=p->next;
      }
      return set;
}
```

OOP练习：链表实现集合类IntSet（续）

```
IntSet IntSet::difference (const IntSet& s) const // 求差集 当前集合-s
{     IntSet set; // 定义拷贝构造函数
      Node *p=head;
      while (p!=NULL)     // while循环遍历单链表
      {     if ( !s.is_element(p->value))
                  set.insert(p->value);
            p=p->next;
      }
      return set;
}
```

练习- 折半查找

一组学生名单存于结构数组中，且已按学号从小到大排序。请设计C++函数，完成用折半法根据学号查找姓名的功能，函数原型为：

char *BiSearchR(Stu stu_array[], int first, int last, int id)

学生信息用如下类型表示：

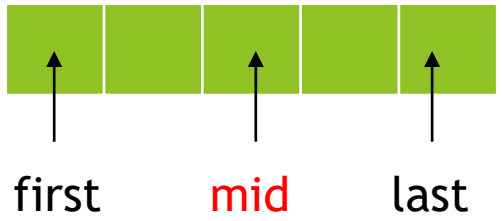
```
struct Stu
{  int id;
   char name[20];
};
```

完成main函数

```
const int N = 50;
char *BiSearchR(Stu stu_array[], int first, int last, int id);
int main( )
{     int num = 0;  Stu stu_a[N];
      for(int i=0;i<50;i++)
          cin>> stu_a[i].id>>stu_a[i].name;
      cout << "Input a student's id:";
      cin >> num;  //从键盘输入待查学生的学号
      char *x = BiSearchR(stu_a, 0, N-1, num);
      if(x == 0)
            cout << "The student's id is error.\n";
      else
            cout << "The student's name is: " << x << endl;
      return 0;
}
```

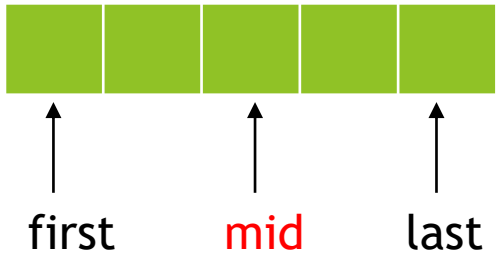
非递归算法

```
char *BiSearchR(Stu stu_array[], int first, int last, int id)
{    int mid;
     while(first<=last)
     {     mid = (first+last)/2;
          if(id == stu_array[mid].id)
               return  stu_array[mid].name;
          if(id >stu_array[mid].id )
               first = mid+1;
          else
               last = mid-1;
     }
     return 0;
}
```



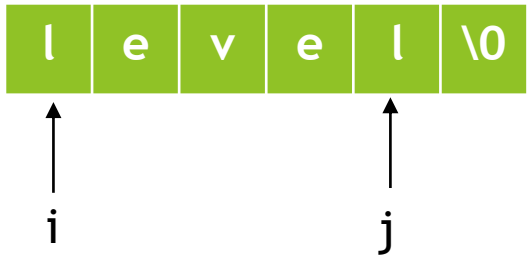
用递归函数实现折半查找

```
char *BiSearchR(Stu stu_array[], int first, int last, int id)
{   if(first > last)
       return 0;
    int mid = (first + last) / 2;
   if(id == stu_array[mid].id)
      return stu_array[mid].name;
   else if(id > stu_array[mid].id)
      return BiSearchR(stu_array, mid + 1, last, id);
   else
      return BiSearchR(stu_array, first, mid - 1, id);
}
```



练习：回文单词判断

- 函数Palindrome（char * str, int i , int j），判断单词是否回文。
- 比如单词： level 就是一个回文单词



- 非递归：

```
for(int i=0, j=length-1; i < j; ++i, --j)
    if(str[i] != str[j])
        return false;
return true;
```

- 递归：分析出递归条件和同质的子问题
  - 当 i<j ：str[i]!=str[j] ,则不是回文
    str[i]==str[j], 则调用Palindrome(str,++i,--j)
  - 当i>=j：是回文；

练习：分别用循环与递归函数实现“台阶问题”

- 一个台阶总共有n级，如果一步可以跳1级，也可以跳2级，求到达第n级台阶的跳法总数。

分析：(1) n=1时，跳法总数为1

(2) n=2时,跳法总数为2

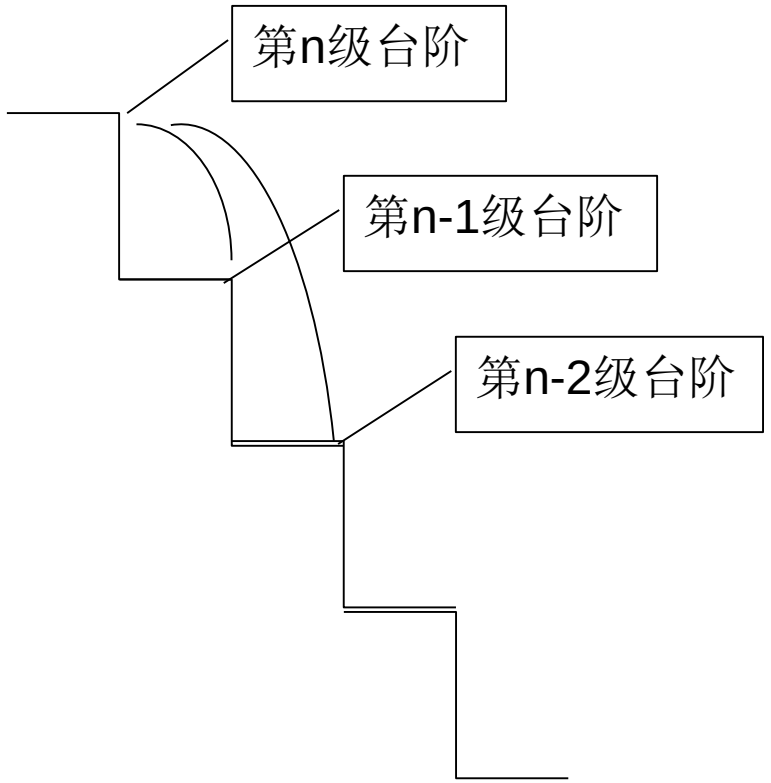
(3) 当n>2时

第n级台阶的跳法总数

= 第n-1级台阶的跳法总数

+

第n-2级台阶的跳法总数



类似斐波拉契数列

台阶问题：循环处理

```
int myStep(int n)
{  if(n==1)
        return 1;
   int step_1 = 1, step_2 = 2;
   for (int i = 3; i <= n; ++i)
   {   int temp = step_1 + step_2;
       step_1 = step_2 ;
       step_2 = temp;
    }
    return step_2;
}
```

台阶问题：递归处理

- 能分析出规律公式，直接翻译成C++语言

```
int myStepR(int n)
{  if(n == 1)
       return 1;
   else if( n == 2)
          return 2;
       else
          return myStepR(n-2) + myStepR(n-1);
}
```

练习—分苹果

- M个苹果放在N个盘子里，允许有的盘子空着不放，问共有多少种不同的放法？说明： 假设3个盘子7个苹果，则5、1、1和1、5、1是同一种放法。

分析：

1、盘子数多于苹果数(N > M)：最多会放满M个盘子，放法和M个盘子时一样 。

2、 盘子数等于或少于苹果数 (N≤M)

(1) 每个盘子都有苹果：每个盘子至少1个苹果，剩下的M-N个苹果，放入N 个盘子中。(子问题)

(2) 如果至少有一个盘子是空的： 将M个苹果放入N-1个盘子中。(子问题)

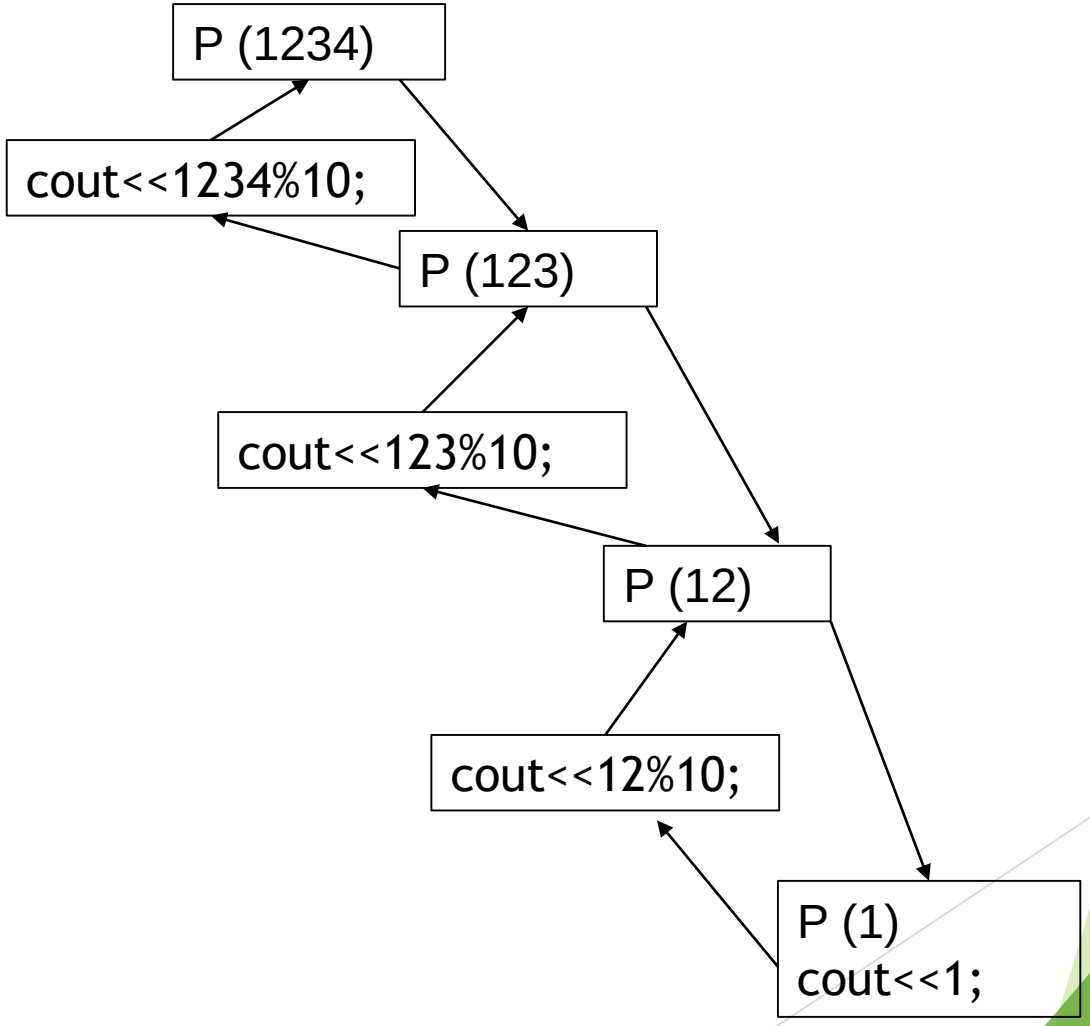
```
int apple(int m, int n)
{  if (m == 0 || n == 1)      return 1;
   if (n > m)                 return apple(m, m);
   if (n<=m)                  return apple(m - n, n) + apple(m, n - 1);
}
```

练习 - 输入为1234时，分析程序的执行结果

```
void print(int n)
{    if ( n >= 10 )
        print( n/10 );
     cout << n%10 ;
}
int main()
{   int n;
    cout << "input n:";
    cin >> n;
    print(n);
    cout << endl;
    return 0;
}
```

input n: 1234

程序执行过程：



练习 - 函数的参数传递

```
void Swap(int x, int *y, int & z)
{      int temp = x;
       x= *y ;
       *y = z ;
       z = temp ;
}

int main( )
{      int i = 1, j = 2 , k = 3;
       Swap(i, &j,k);
       cout<<"i="<<i<<"  j="<<j<<"  k="<<k<<endl;
       return 0;
}
```

练习- 静态局部变量，分析下面程序结果

```
#include <iostream>
using namespace std;
int f()
{    static int s=1;
     s=(7*s+19)%3;
     return s;
}
```

```
int main( )
{ int m;
 m = f( );
 cout<<"1、 m="<<m<<endl;
 m = f( );
 cout<<"2、 m="<<m<<endl;
 m = f( );
 cout<<"3、 m="<<m<<endl;
 return 0;
}
```

练习- 变量的生存期，分析下面程序结果

```
class A
{ int m;
public:
    A(int x=10)
     { m=x;
       cout<<"cos object of m= "<<m<<endl;
     }
    void set(int x) { m=x;}
    A( A &a)
   {  m=a.m;
      cout<<"copy cos"<<endl;
    }
   ~A()
    { cout<<"des object of m= "<<m<<endl;}
};
```

```
A  f( A aa)
{  aa.set(20);
   return aa;
}
A a(40);
int main()
{
  A a1(30);
  A a2(a1);
  A a3;
  a3=f(a2);
  return 0;
}
```

第四章练习- 变量的生存期，分析下面程序结果

cos object of m= 40 → 全局变量 ： A a(40) ；

cos object of m= 30 → main函数中的局部变量： A a1(30);

copy cos → main函数中的局部变量: A a2(a1);

cos object of m= 10 → main函数中的局部变量: A a3;

copy cos → f函数中的局部变量aa: 由实参a2拷贝构造而成

copy cos → return aa; f函数中的局部变量aa拷贝构造临时对象

des object of m= 20 → f函数调用结束，f函数中的局部变量aa 消亡

des object of m= 20 → 完成 a3=f(a2); 临时对象消亡

des object of m= 20 → main函数调用结束，main函数中的局部变量 a3 消亡

des object of m= 30 → main函数调用结束，main函数中的局部变量 a2 消亡

des object of m= 30 → main函数调用结束，main函数中的局部变量 a1 消亡

des object of m= 40 → 程序运行结束，全局变量 a 消亡

有疑问，尽管在QQ上联系我

祝各位同学期末考出好成绩！