20220324-C++练功

1.过程描述

1.1《即兴演讲》

用“你好，最近怎么样”来开启对话。总体而言，即兴演讲的能力还是挺重要的，包括在一些非正式场合如何跟不同的人开启对话，如何清晰传递自己的想法以及阐述自己面临的问题，如何凝练地汇报自己的工作内容及成果等等都对这一能力有不小的需求。一方面得自己内功足够深厚，另一方面也离不开提前准备。

1.2 C++ Prime

For循环 C++ 复制代码

```
对一个数组a[]={1,3,4,5}，可以通过sizeof(a)/sizeof(a[0])来获得数组内元素的个数
--------------------
在循环中，a++表示的是它本身，而++a则是a加1之后的结果。另外注意运算符++的优先级比指针解除引用*要高，注意括号的使用
--------------------
三种反转字符串的方法
string a;
cin >> a;
stringstream ss;
for (int i=0;i<a.size();i++)
{
    ss << a[a.size()-1 - i];
}
cout << ss.str() << endl;

const char* b = "PUNCK";
string c="";
int length = sizeof(b) / sizeof(b[0]);
for (int i = 0; i <length ;i++)
{
    c+=b[length-1 - i];
}
cout << c << endl;

string c = "HKLJ";
char temp;
int i, j;
for (i = 0, j = c.size()-1; i<j;++i,--j) {
    temp = c[i];
    c[i] = c[j];
    c[j] = temp;
}
cout << c;

BTW：创建字符串可以用string A="zifuchuan",char* A="zifuchuan"（通常会要求用const char*）,char A[]="zifuchuan"
-------------------------
数组名是数组的地址，用引号括起来的字符串常量也是其地址。下面的关系表达式判断的是两个字符串是否存储在相同的地址上，答案是否定的。可以用strcmp进行比较
```

While循环 C++ 复制代码

```
为类型创建别名的两种方式
1)宏。宏本质上只是复制粘贴替换，当在声明一系列变量时，可能会出现问题
#define FLOAT_POINTER float*
FLOAT_POINTER a,b
这样定义的a是指针，而b是float
2) typedef
typedef本质上是为已有的类型创建一个新名称
typedef float* FLOAT_POINTER
FLOAT_POINTER a,b
a,b都为指针
-------------------------------------
延时循环+clock的使用
#include <ctime>
int secs = 3;
clock_t delay = secs * CLOCKS_PER_SEC; //转换为以系统时间单位为单位
clock_t start = clock();
int i = 0;
while (clock() - start < delay);
cout << "done" << endl;
-------------------------------------
```

输入&二维数组 C++ 复制代码

```
cin会忽略空格和换行符，可以使用cin.get()
char ch;
cin.get(ch);
int count=0;
while (ch != '#')
{
    cout << ch;
    count++;
    cin.get(ch);
}
cout << endl << count << endl;
-----------------------------------------
二维数组的形式比较简单
int a[2][2] = {
    {2,3},
    {4,5}
};
```

条件

C++ | 复制代码

```
可以使用? 来代替简单的if-else语句
3>5?1:2,返回2
----------------------------------------
switch-case示例
int choice;
cin>>choice;
switch(choice)
{
    case 1:cout<<"hello";break;
    case 2:cout<<"good";break;
    case 3:cout<<"OK";break;
    default:cout<<"byby";
}
注意加break，否则会把后面的case都执行了
```

文件IO C++ 复制代码

```cpp
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
写入文件
//新建一个文本，并往其中写入
//创建好ofstream对象之后便可以像cout一样使用它
ofstream outfile;
outfile.open("helloworld.txt");
outfile << fixed;//表示用普通的方式输出浮点数
outfile.precision(2);//控制浮点数的精度
float a = 35.679f;
float b = 89.787f;
outfile << "your weight is " << a << endl;
outfile << "your height is " << b << endl;
outfile.close();
-----------------------------------------
读取文件
ifstream infile;
char filename[50];
cout << "enter your file name: ";
cin.getline(filename, 50);
infile.open(filename);
if (!infile.is_open())
{
    exit(EXIT_FAILURE);
}
double value;
double sum = 0.0;
int count = 0;
infile >> value;
while (infile.good())//good指出最后一次读取输入的操作是否成功
{
    ++count;
    sum += value;
    infile >> value;
}
/*这种形式更简单，需要把上面的infile>>value删了
double value;
double sum = 0.0;
int count = 0;
while (infile>>value)
    {
        ++count;
        sum += value;
```

```
46          }
47      */
48      if (infile.eof())
49      {
50          cout << "End of file" << endl;
51      }
52      else if (infile.fail())
53      {
54          cout << "inputs terminated by data dismatch" << endl;
55      }
56      else
57      {
58          cout << "inputs terminated by unknown reason.\n";
59      }
60      if (count == 0)
61      {
62          cout << "no data.\n";
63      }
64      else
65      {
66          cout << "items read: " << count << endl;
67          cout << "sum: " << sum << endl;
68      }
69      infile.close();
70      return 0;
71
```

函数 C++ 复制代码

main（成为调用函数）在调用函数时，如果函数定义时未声明引用，则逻辑是这样的：
int func(int x);
1.被调用函数创建一个x变量，并将实参的值复制一份放到这个临时变量里面；
2.被调用函数执行函数体，将返回值放到寄存器里，临时变量销毁；
3.main函数找到这个返回值，并在内部进行使用
在这个过程中，实参不实际参与被调用函数的执行过程，因此也不会受到影响。在声明引用时才会根据被调用函数的执行逻辑发生变动

C++中数组名是第一个元素的地址，即cookies==&cookies[0]，但有一些例外：
1.数组声明时利用数组名来标记存储位置
int cookies[]={1,2,3}
2.对数组名使用sizeof来得到整个数组的长度
3.将地址符应用于数组名时，将返回整个数组的地址
在c++中，当且仅当用于函数头或函数原型时，int* arr和int arr[]的含义是相同的。但无论arr是指针还是数组名，用方括号来访问数组元素都是允许的。
两个重要等式：
arr[i]=*(arr+i)
&arr[i]=arr+i
将指针加1，实际上是加了一个与指针指向的类型的长度相等的值（多少多少字节）

将数组作为参数时，有几个东西传入被调用函数：
1.数组的位置（地址）
2.包含的元素的类型
3.元素数目
利用这些信息函数可以使用实参数组。上面提到，传递常规变量时，函数将使用该变量的拷贝，而传递数组时，函数将使用实参（并不矛盾，相当于创建了一个新变量来存放第一个元素地址的副本）

```
int func(int arr[], int size) //加上const可以保护数组，但这种保护只是指改变元素的行为将报错
{
    for (int i = 0; i < size; i++)
    {
        arr[i] += 1;
    }
    return arr[0];
}

int main()
{
    const int size = 4;
    int demo[size] = {1,2,3,4};
    int result = func(demo, size);
    cout << result << endl;//输出2
    cout << demo[0] << endl;//也输出2
}
```

```
可以看到，实参也被改变了。
-----------------------------------------------
int a=10;
int* ptr = &a;//如果这里用const的话，则下面一行是不行的
*ptr = 20;
cout << a << endl;//a变成了20

int a=10;
const int* ptr=&a;
int b=20;
ptr=&b;//现在ptr指向20
上面的const只能防止修改ptr指向的值，但不能防止修改ptr的值

int a=10;
int* const ptr=&a;
*ptr=20;
const的位置变了，现在ptr的值不能修改，但允许修改ptr指向的值

还可以前后都用const，来个两面夹击，按的死死的
-----------------------------------------
二维数组的声明
int sum(int (*arr)[2], int size);//arr是指针而不是数组，指向由2个int组成的数组
//另一种形式
//int sum(int arr[][2],int size)
int main()
{
    int a[3][2] = {
        {2,4},
        {5.6},
        {4,5}
    };
    int result = sum(a, 3);
}
-----------------------------------------
字符串的名也是表示第一个字符的地址
unsigned int count(const char* demo, char character)
{
    int num = 0;
    while (*demo) //当检查到结束字符就结束
    {
        if (*demo==character)
            num++;
        demo++;
    }
    return num;
}
int main()
{
```

```
    const char* name = "Willianwu";
    char testCharacter = 'i';
    cout << count(name, testCharacter) << endl;
}
char* demo和char[]的区别在于前者有内置的结束字符，而后者如果不包含结束字符的话只是数组而不是字符串
----------------------------------------
利用new创建动态数组，记得考虑最后的控制
char* dupli(char c, int n)
{
    char* ptr=new char(n+1);
    ptr[n] = '\0';
    while (--n>=0)
```

递归 C++ 复制代码

```
递归在函数中的表现形式便是在一个函数的函数体内部包含了对这个函数的调用
例子：一个递归调用
void CountDown(int n)
{
    cout << n << endl;
    if (n > 0)
    {
        CountDown(n - 1);
    }
}
int main()
{
    CountDown(5);
    return 0;
}

```

2.结果输出

prime啃起来比预期慢很多，主要有很多比较细致的点，直接翻过去感觉有点可惜，还是得在程序里面试验一下才比较有底。今天本来准备把函数部分看完，看来是没戏了，眼睛不允许。今天倒也基本没浪费一点时间，明天接着加油吧。