C++机试基础知识准备

C++语言特性

- new二维数组

```
void TestFunc_pointer(unsigned int height, unsigned int width)
{
    int i, j;
    //  数组的元素类型为‘int *’，可赋值给指向‘int *’的指针.
    int **array2D = new int *[height];
    for(i=0; i<height; ++i)
    {
        array2D[i] = new int[width];
    }
    // 首先回收低一级的动态数组.
    for(i=0; i<height; ++i)
    {
        delete[] array2D[i];
    }
    // 然后回收高一级的动态数组.
    delete[] array2D;
}
```

- 传值、传引用、传指针
- 内联函数（inline）
- 默认参数
- 枚举（enum）

```
enum Roster {Tom, Sharon, Bill, Teresa, John};
Roster student;
student = Sharon;
```

- 模板

```
//函数模板
template <typename T>
void swap( T& v1, T& v2)
{
    T temp;
    temp = v1;
    v1 = v2;
    v2 = temp;
}

//类模板
template <typename ElementType> //genericStack.h
class Stack {
public:
    Stack();
    ~Stack();
```

```
    void push(ElementType obj) throw(std::bad_alloc);
    void pop() throw(std::logic_error);
    ElementType getTop() const throw(std::logic_error);
    bool isEmpty() const;
private:
    struct Node // 栈结点类型
    {
        ElementType element; // 结点中存放的元素
        Node* next; // 指向下一结点的指针
    };
    Node* top; // 栈顶
};

template <typename ElementType>
void Stack<ElementType>::push( ElementType obj ) throw(std::bad_alloc)
{
    Node* temp;
    try {
        temp = new Node;
        temp -> element = obj;
        temp -> next = top;
        top = temp;
    }
    catch (std::bad_alloc e) { // 内存分配失败时进行异常处理
        throw; // 重新抛出异常
    }
}

//调用方式
Stack<int> stack;
```

类相关知识

- 基本类结构

```
#ifndef DATE_H
#define DATE_H
class DATE // DATE.h----Specification file of class DATE
{
    public:
        DATE( int, int, int ); //构造函数
        DATE(); //缺省构造函数
        DATE(const DATE& obj); //拷贝构造函数
        DATE& operator=(const DATE& other); //有时需要重载赋值符号来完成深拷贝
        void Set( int, int, int );
        int getMonth() const;
        int getDay() const;
        int getYear() const;
        void Print() const;
        void Increment();
        void Decrement();
        ~DATE(); //析构函数
    private:
        int month, day, year;
};
#endif
```

- 类中有指针注意深拷贝和浅拷贝的问题
- this指针使用
- 类常量成员

```
class Demo {
    public:
    Demo( ): data1(0) // 常量数据成员只能在构造函数初始化列表中初始化
    {
        // data1 = 0; // 此处不能对常量数据成员data1赋值
        data = 0;
    }
    private:
        int data; // 一般的数据成员
        const int data1; // 常量数据成员
};
```

- 静态成员
    - 静态成员函数不属于任何对象
    - 静态成员函数没有this指针
    - 静态成员函数不能直接访问类的非静态数据成员，只能直接访问类的静态数据成员
    - 静态成员变量需在类外面定义一次

```
class DATE // DATE.h
{
public:
    DATE( int =2000, int =1, int = 1);
    static void getCount( );
private:
    int month, day, year;
    static int count;
};

//DATE.cpp StaticMember
int DATE::count = 0; //必须在类定义体的外部再定义一次
DATE::DATE( int initYear, int initMonth, int initDay )
{
    year = initYear;
    month = initMonth;
    day = initDay;
    count++;
}
void DATE::getCount()
{
    cout << "There are " << count << " objects now" << endl;
}
```

- 组合

```
#include"data.h"
class Employee
{
public:
    Employee( char *, char *, int, int, int, int, int, int );
    void print() const;
    ~Employee();
```

```
private:
    char firstName[ 25 ];
    char lastName[ 25 ];
    Date birthDate;
    Date hireDate;
};
Employee::Employee( char *fname, char *lname, //Empoyee.cpp
int bmonth, int bday, int byear,
int hmonth, int hday, int hyear )
: hireDate( hmonth, hday, hyear ),  //尤其注意这两行
birthDate( bmonth, bday, byear )    //
{ }
```

- 友元重载VS直接重载

```
INTEGER operator + (INTEGER& left, INTEGER& right)
{
    INTEGER temp;
    temp.value = left.value + right.value;
    return temp;
}
INTEGER operator +(INTEGER& other) // 重载加法运算符
{
    INTEGER temp;
    temp.value = value + other.value;
    return temp;
}
```

- 继承

```
// SPECIFICATION FILE ( time.h )
class Time
{
public :
    void Set ( int hours , int minutes , int seconds ) ;
    void Increment ( ) ;
    void Write ( ) const ;
    Time ( int initHrs, int initMins, int initSecs ) ; // constructor
    Time ( ) ; // default constructor
private :
    int hrs, mins, secs ;
} ;

// SPECIFICATION FILE ( exttime.h)
#include “time.h”
enum ZoneType {EST, CST, MST, PST, EDT, CDT, MDT, PDT } ;
class ExtTime : public Time // Time is the base class,默认是private
{
public :
    ExtTime(int initHrs, int initMins, int initSecs, ZoneType initZone);
    ExtTime ( ) ; // default constructor
    void Set ( int hours, int minutes, int seconds, ZoneType timeZone) ;
    void Write ( ) const ;
private :
    ZoneType zone ; // added data member
} ;
```

```
ExtTime::ExtTime(int initHrs,
                 int initMins,
                 int initSecs,
                 ZoneType initZone )
                 : Time(initHrs, initMins, initSecs) //基类构造函数在这里调用
{
    zone = initZone;
}
```

- public、private、protected
  - 类的一个特征就是封装，public和private作用就是实现这一目的。所以 用户代码（类外）可以访问public成员而不能访问private成员；private成员只能由类成员（类内）和友元访问。
  - 类的另一个特征就是继承，protected的作用就是实现这一目的。所以protected成员可以被派生类对象访问，不能被用户代码（类外）访问。
  - “私有”和“不可访问”有区别：私有成员可以由派生类本身访问，不可访问成员即使是派生类本身也不能访问。

基类中成员的访问控制	继承访问控制	派生类中继承成员的访问控制
public	public	public
protected		protected
private		不可访问
public	protected	protected
protected		protected
private		不可访问
public	private	private
protected		private
private		不可访问

- 继承中的构造函数问题
  - 基类的构造函数不被继承，派生类中需要声明自己的构造函数。
  - 派生类的构造函数中只需要对本类中新增成员进行初始化即可。对继承来的基类成员的初始化是通过自动调用基类构造函数完成的。
  - 派生类的构造函数需要给基类的构造函数传递参数。
  - 构造函数的调用次序： 基类->本类中成员变量->本类自己的构造函数。
  - 析构函数调用顺序：与构造函数相反。
- 虚基类（virtual）：普通基类与虚基类之间的唯一区别只有在派生类重复继承了某一基类时才表现出来。
- 虚函数：与继承相结合以实现运行时多态性。在公有继承层次中的一个或多个派生类中对虚函数进行重定义，然后通过指向基类的指针（或引用）调用虚函数来实现实现运行时多态性。
  - 若函数不是虚函数，形参类型决定调用哪个函数（静态绑定）
  - 若函数是虚函数，实参类型决定调用哪个函数（动态绑定）
  - 用虚函数实现动态绑定的关键：必须用基类指针（或基类引用）来访问虚函数。
  - 在派生类中**重定义**从基类中继承过来的虚函数(函数原型保持不变)，该重定义的函数在该派生类中仍是虚函数。
  - 函数**重载**，虚特性丢失。
- 纯虚函数：virtual 返回值类型 函数名(形参表) =0 ;

机试注意

- 格式化输出

```
int printf(const char *format, ...)
```

format的格式：%[flags][width][.precision][length]specifier

flags（标识）	描述
-	在给定的字段宽度内左对齐，默认是右对齐（参见 width 子说明符）。
+	强制在结果之前显示加号或减号（+ 或 -），即正数前面会显示 + 号。默认情况下，只有负数前面会显示一个 - 号。
空格	如果没有写入任何符号，则在该值前面插入一个空格。
#	与 o、x 或 X 说明符一起使用时，非零值前面会分别显示 0、0x 或 0X。 与 e、E 和 f 一起使用时，会强制输出包含一个小数点，即使后边没有数字时也会显示小数点。默认情况下，如果后边没有数字时候，不会显示显示小数点。 与 g 或 G 一起使用时，结果与使用 e 或 E 时相同，但是尾部的零不会被移除。
0	在指定填充 padding 的数字左边放置零（0），而不是空格（参见 width 子说明符）。

width（宽度）	描述
(number)	要输出的字符的最小数目。如果输出的值短于该数，结果会用空格填充。如果输出的值长于该数，结果不会被截断。
*	宽度在 format 字符串中未指定，但是会作为附加整数值参数放置于要被格式化的参数之前。

.precision（精度）	描述
.number	对于整数说明符（d、i、o、u、x、X）：precision 指定了要写入的数字的最小位数。如果写入的值短于该数，结果会用前导零来填充。如果写入的值长于该数，结果不会被截断。精度为 0 意味着不写入任何字符。 对于 e、E 和 f 说明符：要在小数点后输出的小数位数。 对于 g 和 G 说明符：要输出的最大有效位数。 对于 s: 要输出的最大字符数。默认情况下，所有字符都会被输出，直到遇到末尾的空字符。 对于 c 类型：没有任何影响。 当未指定任何精度时，默认为 1。如果指定时不带有一个显式值，则假定为 0。
.*	精度在 format 字符串中未指定，但是会作为附加整数值参数放置于要被格式化的参数之前。

- getchar() & putchar()
- gets() & puts()
- STL
  - vector: empty(), pop_back(), erase(a.begin()+1,a.begin()+3), push_back(), insert(a.begin()+1,5), size()
  - pair: make_pair(), p.first, p.second, #include <utility>
  - stack: empty(), push(), pop(), size(), top()
  - queue: empty(), push(), pop(), size(), front(), back()
  - priority_queue:
    - priority_queue <int,vector<int>,greater<int> > q;
    - priority_queue <int,vector<int>,less<int> >q;
    - pair的比较，先比较第一个元素，第一个相等比较第二个
    - 自定义比较

```
struct cmp
{
    bool operator()(node a,node b)
    {
        if(a.x == b.x)  return a.y >= b.y;
        else return a.x > b.x;
    }
};
priority_queue<node,vector<node>,cmp> pq;
```

    - unordered_map: insert(), find(), unordered_map<string, string> umap;
    - algorithm:
        - sort(a.begin(),a.end())
        - reverse(a.begin(), a.end())
- string
    - c_str(), length(), to_string(int)
- char
    - strcpy(p, p1) 复制字符串
    - strcat(p, p1) 附加字符串
    - strlen(p) 取字符串长度
    - strcmp(p, p1) 比较字符串
    - atoi(p) 字符串转换到 int 整型
    - int substr(char dst[], char src[],int start, int len)