华为C++机试

String

1.string 与 int 转换

- **int 转 string**

```
to_string()
string = int + ""
```

- **string 转 int**

```
//1.
int num = atoi(str.c_str());  //首字符为数字则返回数字，非数字则返回0，atoi的参数必须为const char*，另
//2.
int num = str-'0';
```

2.string类型的字符串长度获取的三种方法

```
size(),length(),strlen()
```

3.输入含空格的字符串

```
scanf("%[a-z A-Z0-9]",str)
getline(cin, str)  //包含头文件<string>
cin.get (char *str, int maxnum)
cin.getline (char *str, int maxnum)
```

4.string划分与删除字符md5()kv

- 划分字符串（按';'）

```
string str;
cin>>str;
int n = str.size();
for (int i = 0; i < n; ++i){
    if (str[i] == ';'){
        str[i] = ' ';
    }
}
istringstream out(str);  //初始化out为str
string mov[n];
int temp  = 0;
while (out >> mov[temp])
    temp++;
```

- 获取字符串的一段

```
string a = s.substr(0,5);  //获得字符串s中从第0位开始的长度为5的字符串
```

- 删除某个字符

```
string str;
string target;
int pos = str.find(target);
n = target.size();
str = str.erase(pos,n);  //从pos这个位置开始删除n个字符，后边的字符自动向前覆盖
```

5.判断字符串中是否含有某个子串

```
idx=a.find(b);  //如果存在，返回起始位置。不存在则返回 string::npos
strstr(a.c_str(), b.c_str())  //char * strstr(const char *str1, const char *str2);
```

6.查找字符在字符串中首次出现的位置

```
//strchr() 用来查找某字符在字符串中首次出现的位置，其原型为：
char * strchr (const char *str, char c);
```

7.字符串拼接

```
strcat(str,ptr);  //将字符串ptr内容连接到字符串str后
```

New的使用

使用new建立动态数组

```
int num;
cin>>num;
int *index = new int[num];

//用new创建一个二维数组
//1.
int (*p)[line] = new int[row][line];
//删除二维数组：
delete []p;
int **p

//2.
p = new int*[row];
for(int i = 0;i < row; i++)
  p[i] = new int[line];
//删除二维数组
for(int i=0;i<row;i++)
  delete [] p[i];
delete [] p;
```

STL

1. 迭代器

```
vector<int>::iterator iter=ivec.bengin();  //将迭代器iter初始化为指向ivec容器的第一个元素
```

2. vector

2.1 创建vector对象

```
  vector<int> vec;  //声明一个int型向量
  vector<int> vec(8); //声明一个初始大小为8的int型向量
  vector<int> vec(10, -1) //声明一个初始大小为10且值都是-1的int型向量
```

2.2 vector赋值与修改

```
//赋值
  vec.assign(v1.begin(), v1.begin()+8);  //将动态数组v1的[v1.begin,v1.begin()+8)赋值给动态数组vec
  //修改
  auto &val = vec.back(); //val为指向最后一个元素的引用
  val = 2;  //vector需要使用引用修改，不可以使用数组下标直接修改
```

2.3 vector访问

```
//通过下标访问、修改向量元素，但不可以通过下标添加元素
for(int i = 0; i < vec.size(); i++){
    cout << vec[i];
}
//通过迭代器访问数组
for(vector<int>::iterator iter = vec.begin(); iter != vec.end(); iter++){
    cout << *iter;
}
//通过指针访问数组
int *p = vec.data();
for (int i = 0; i < 5; i++){
  cout << *p++;
}
```

2.4 添加元素

```
vec.push_back(p);  //在数组vec的末尾添加元素p
vec.insert(v.it,p);  //在数组vec.it指向位置插入元素p
vec.insert(v.it,n,p);  //在数组vec.it指向位置插入n个元素p
```

2.5 删除元素与删除重复元素

```
v.pop_back();  //删除数组v末尾的元素
v.erase(v.it);  //删除数组v.it指向位置的元素
//删除重复元素
sort(vec.begin(), vec.end());  //unique只移除相邻的重复元素所以需要先排序
vector<int>::iterator ite = unique(vec.begin(), vec.end());  //unique将重复的元素移到末尾，返回末尾
vec.erase(ite, vec.end());  //删除残余元素
```

2.6 反转reverse()

```
reverse(beg,end);
reverse_copy(sourceBeg,sourceEnd,destBeg);
//reverse()会将区间[beg,end)内的元素全部逆序;
//reverse_copy()会将源区间[sourceBeg,sourceEnd)内的元素复制到
//"以destBeg起始的目标区间"，并在复制过程中颠倒安置次序.
```

2.7 排序

```
vector<int> v{ 0, 3, 5, 7, 8, 1, 4, 9, 2, 6};
//默认情况下升序排列
sort(v.begin(), v.end()); // v{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
//可自定义排序方式（此为降序）
bool Comp(const int &a, const int &b){
  return a>b;
}
sort(v.begin(), v.end(), Comp); // v{ 9, 8, 7, 6, 5, 4, 3, 2, 1, 0}
//亦可函数内部自定义
sort(v.begin(), v.end(), [](int &a, int &b){return a>b;});  // v{ 9, 8, 7, 6, 5, 4, 3, 2, 1, 0}
//sort给数组a[20]排序sort(a, a+20);
```

3. map

3.1 pair类型的定义和初始化

pair类型包含了两个数据值，通常有以下的一些定义和初始化的一些方法：

```
pair<T1, T2> p; ：  定义了一个空的pair对象p，T1和T2的成员都进行了值初始化
pair<T1, T2> p(v1, v2); ：  p是一个成员类型为T1和T2的pair；first和second成员分别用v1和v2进行初始化。
pair<T1, T2> p = {v1, v2} ：等价于p(v1, v2)
make_pair(v1, v2) ：  以v1和v2值创建的一个新的pair对象
```

3.2 创建map对象

map是键-值对的组合，即map的元素是pair，其有以下的一些定义的方法：

```
map<k, v> m;         ：  定义了一个名为m的空的map对象
map<k, v> m2(m); ：  创建了m的副本m2
map<k, v> m3(m.begin(), m.end()); ：  创建了map对象m3，并且存储迭代器范围内的所有元素的副本
```

3.3 map元素访问

```
- mymap['a'] = "an element";
- mymap.at('a') = "an element";
```

3.4 map中元素的插入

- 3.4.1 使用下标[]插入
```
mymap[0] = 'a';
```
- 3.4.2 使用insert()插入元素
```
// （1）插入单个值
mymap.insert(std::pair<char, int>('a', 100));
// （2）指定位置插入
std::map<char, int>::iterator it = mymap.begin();
mymap.insert(it, std::pair<char, int>('b', 300));
// （3）范围多值插入
std::map<char, int> anothermap;
anothermap.insert(mymap.begin(), mymap.find('c'));
```

3.5 map删除元素

```
// erase() 删除元素
mymap.erase(0);                    //  (1) 删除key为0的元素
mymap.erase(mymap.begin());  //  (2) 删除迭代器指向的位置元素
// 查找关键字1在容器map中出现的次数，如果不存在则为0
mymap.count(1);
// find()若存在，返回指向该key的迭代器
// 若不存在，则返回迭代器的尾指针，即 mymap.end()，即 -1
map<int, int>::iterator it_find;
it_find = mp.find(0);
//equal_range() 返回一个迭代器pair，表示关键字 == k的元素的范围。
//若k不存在，pair的两个成员均等于c.end()
//lower_bound()/upper_bound()函数需要加载头文件#include<algorithm>,
//其基本用途是查找有序区间中第一个大于或等于/大于某给定值的元素的位置，
//其中排序规则可以通过二元关系来表示。
//rbegin() 返回一个指向map尾部的逆向迭代器
//rend() 返回一个指向map头部的逆向迭代器
//key_comp() 比较key_type值大小
//value_comp() 比较value_type值大小
```

3.6 两种遍历方式

```
for (map<int, char>::iterator iter = mymap.begin(); iter != mymap.end(); iter++)
cout << iter->first << " ==> " << iter->second << endl;
```

```
    for (auto& x : mymap) {
        std::cout << x.first << ": " << x.second << '\n';
    }
```

3.7 map排序

3.7.1 map转vector

```
vector<pair<int, float> > map_vec(map.begin(), map.end());
```

3.7.2 对vector排序

```
    sort(vect.begin(), vect.end());
```

4. stack&queue

```
push();  //进容器
pop();  //出容器，没有返回值
//取栈顶/对头元素，并没有删除元素
top()/front();
my2.swap(mystack);  //swap函数可以交换两个栈的元素
```

经典问题

动态规划

- 经典动态规划

回溯算法

贪心算法

分治算法

分支界限算法

01背包问题

面试经验

const指针

```
const double pi = 3.14; // pi是一个常量，不能改变它的值

const double *cptr = &pi; //cptr指向pi,注意这里的const不能丢，因为普通指针不能指向常量对象，即，不能用

*cptr = 3.33;  //错误,试图改变所指对象的值。不能改变指针所指对象的值

cout << cptr << endl; //输出cptr的值Kv

//虽然不能改变其所指对象的值，但是它可以指向别的常量对象
//这样的话 指针的值（也就是存放在指针中的那个地址）也会改变
```

static

- 静态变量：
  static sta; //在函数中声明则不会随函数调用结束而销毁
- 静态函数：
  1.静态函数不能被其它文件所用；
  2.其它文件中可以定义相同名字的函数，不会发生冲突；
- 静态数据成员：
    用于修饰 class 的数据成员，即所谓“静态成员”。这种数据成员的生存期大于 class 的对象（实体 instance）。
    static只会被初始化一次，于实例无关。
- 静态成员函数：
    因为静态是属于类的，所以静态成员函数不能访问非静态(包括成员函数和数据成员)，但是非静态可以访问静态

volatile

volatile关键字是一种限定符用来声明一个对象在程序中可以被语句外的东西修改,比如操作系统、硬件或并发执行线程。
遇到该关键字，编译器不再对该变量的代码进行优化，不再从寄存器中读取变量的值，而是直接从它所在的内存中读取值，
当两个线程都要用到某一个变量且该变量的值会被改变时，应该用volatile声明，该关键字的作用是防止优化编译器把变量

堆栈

```
//对于一个完整的程序，在内存中的分布情况如下图：
1.栈区：  由编译器自动分配释放，像局部变量，函数参数，都是在栈区。会随着作用于退出而释放空间。
2.堆区：程序员分配并释放的区域，像malloc(c),new(c++)
3.全局数据区(静态区)：全局变量和静态便令的存储是放在一块的，初始化的全局变量和静态变量在一块区域，未初始化的
//栈溢出问题
可以使用命令调整栈的大小改善栈溢出问题。
#pragma comment(linker,"/STACK:102400000,1024000")  //将堆栈改变为100M
第一个值是堆栈的保留空间，第二个值是堆栈开始时提交的物理内存大小。
```

内联函数与宏

函数有一个潜在的缺点：调用函数比求解等价表达式要慢得多。在大多数的机器上，调用函数都要做很多工作：调用前要先
C++中支持内联函数，其目的是为了提高函数的执行效率，用关键字 inline 放在函数定义(注意是定义而非声明，下文继续

```
inline int max(int a, int b){
    return a > b ? a : b;
}
```

则调用： cout << max(a, b) << endl;
在编译时展开为： cout << (a > b ? a : b) << endl; 从而消除了把 max写成函数的额外执行开销。

- 内联函数和宏：
  1、宏容易出错；
  2、宏不可调试；
  3、宏无法操作类的私有对象；
  4、内联函数可以更加深入的优化；

- 定义在类声明之中的成员函数将自动地成为内联函数。

- 内联函数的调用要复制代码，将使程序的总代码量增大，消耗更多的内存空间。以下情况不宜使用内联：
  （1）如果函数体内的代码比较长，使用内联将导致内存消耗代价较高。
  （2）如果函数体内出现循环，那么执行函数体内代码的时间要比函数调用的开销大。

多态、重载、重写、动态绑定与虚机制

只有虚函数才使用的是动态绑定，其他的全部是静态绑定。

```
class B{
    void DoSomething();
    virtual void vfun();
}
class C : public B{
    void DoSomething();//首先说明一下，这个子类重新定义了父类的no-virtual函数，这是一个不好的设计，会导
    virtual void vfun();
}
class D : public B{
    void DoSomething();
    virtual void vfun();
}
D* pD = new D();
B* pB = pD;
```

- 让我们看一下，pD->DoSomething()和pB->DoSomething()调用的是同一个函数吗?
不是的，虽然pD和pB都指向同一个对象。因为函数DoSomething是一个no-virtual函数，它是静态绑定的，也就是编译器

- 让我们再来看一下，pD->vfun()和pB->vfun()调用的是同一个函数吗?
是的。因为vfun是一个虚函数，它动态绑定的，也就是说它绑定的是对象的动态类型，pB和pD虽然静态类型不同，但是他们

- 在继承中要构成多态还需要两个条件:
  a．调用函数的对象必须是指针或者引用。
  b．被调用的函数必须是虚函数，且完成了虚函数的重写。

- 纯虚函数就是在虚函数后加了“=0”,而且在基类中没有任何实现，那么包含纯虚函数的类自然就被称为抽象类，抽象类是