day\_1

1.在（）情况下适宜采用 inline 定义内联函数

A 函数体含有循环语句 B 函数体含有递归语句 C 函数代码少、频繁调用 D 函数代码多，不常调用

错选 D------正确答案C

解析:内联函数的使用场景：（1）函数本身内容比较少，代码比较短，函数功能简单

                                                 （2）函数被调用的频繁，比如循环中的函数

         内联函数的利与弊：利：避免了指令来回跳转；弊：代码被复制了多次，增加了代码量，占用更多空间

         不适用的场景：函数代码多，功能复杂，体积庞大；递归函数

2.在 c++ 语言中，对函数参数默认值描述正确的是（）

 A 函数参数的默认值只能设定一个

 B 一个函数的参数若有多个，则参数默认值的设定可以不连续

 C 函数参数必须设定默认值

 D 在设定了参数的默认值后，该参数后面定义的所有参数都必须设定默认值

解析：对于A选项来说，函数参数的默认值可以设置多个；对于B选项，函数参数若有多个时，设置规则要么是全缺省，要么是半缺省。对于带参数的函数，设置默认值的方向必须是从左到右。对于C选项，函数参数在不设置默认参数时，默认初始化为0；对与选项D：在设定了参数的默认值以后，后面的参数必须设定默认值，否则编译器无法判断在调用的时候是否有参数遗漏。

【组队竞赛】牛牛举办了一次编程比赛,参加比赛的有3\*n个选手,每个选手都有一个水平值a\_i.现在要将这些选手进行组队,一共组成n个队伍,即每个队伍3人.牛牛发现 队伍的水平值等于该队伍队员中第二高水平值。

 例如: 一个队伍三个队员的水平值分别是3,3,3.那么队伍的水平值是3

           一个队伍三个队员的水平值分别是3,2,3.那么队伍的水平值是3

           一个队伍三个队员的水平值分别是1,5,2.那么队伍的水平值是2

           为了让比赛更有看点,牛牛想安排队伍使所有队伍的水平值总和最大。

           如样例所示: 如果牛牛把6个队员划分到两个队伍

           如果方案为: team1:{1,2,5}, team2:{5,5,8}, 这时候水平值总和为7.

           而如果方案为: team1:{2,5,8}, team2:{1,5,5}, 这时候水平值总和为10.

           没有比总和为10更大的方案,所以输出10.

#include <vector>

#include <algorithm>

#include <iostream>

using namespace std;

int main()

{

    int n = 0;

    while (cin >> n)

    {

        vector<int>nums(3 \* n);

        long max = 0;

        for (int i = 0; i <nums.size();++i)

            cin>>nums[i];

        sort(nums.begin(),nums.end());

        for (int i = n;i <= nums.size() - 2;i += 2)

            max += nums[i];

        cout<<max<<endl;

    }

}

【删除公共字符】输入两个字符串，从第一字符串中删除第二个字符串中所有的字符。例如，输入”They are students.”和”aeiou”，则删除之后的第一个字符 串变成”Thy r stdnts.”

//在第一个字符串中删除第二个字符串中出现的所有字母

#include<iostream>

#include <algorithm>

using namespace std;

class Solution

{

public:

    void delect\_str(string &str1,string &str2)

    {

        if (str1.empty() || str2.empty())

            return;

        for (int i = 0;i < str1.size();++i)

        {

            for (int j = 0;j < str2.size();++j)

            {

                int pos = str1.find(str2[j]);

                if (pos == -1)

                    continue;

                str1.erase(pos,1);

            }

        }

    }

};

int main()

{

    std::string str1;

    std::string str2;

    std::getline(std::cin,str1);

    std::getline(std::cin,str2);

    Solution S;

    S.delect\_str(str1,str2);

    cout<<str1<<endl;

}

day\_2

1.在32位cpu上选择缺省对齐的情况下，有如下结构体定义：

struct A{

unsigned a : 19;

unsigned b : 11;

unsigned c : 4;

unsigned d : 29;

char index; };

则sizeof（struct A）的值为（）

解析:这道题主要考察位段的理解，首先19表示在32位比特中，占了19个比特，19 + 11 = 30bite，在存放下一个unsigned c : 4的时候空间不够了，所以重新开辟了一个32bite的空间用来存放c，同理，放入29的时候也不够用了，继续开辟。所以内存里是这样存放这几个数据的：

19 + 11

4

29

8

共4个32bite的内存，所以4 \* 32 / 8 = 16；

2.关于c++的inline关键字,以下说法正确的是()

A.使用inline关键字的函数会被编译器在调用处展开

B.头文件中可以包含inline函数的声明

C.可以在同一个项目的不同源文件内定义函数名相同但实现不同的inline函数

D.定义在Class声明内的成员函数默认是inline函数

E.优先使用Class声明内定义的inline函数

F.优先使用Class实现的内inline函数的实现

解析:

A:错误，内联函数只是程序员希望它成为内联函数，但是编译器有权拒绝这个请求，比如当函数体过大的时候

B:错误，头文件中不仅要包含inline函数的声明，而且必须包含定义，且在定义时候必须加上inline

C:错误，inline函数可以定义在源文件中，但是多个源文件中的同名inline函数的实现必须相同，一般来说inline函数的定义放在头文件中更加合适

D:正确

EF:错误，不管在class声明中定义的inline函数，还是在class实现中定义的inline函数，不存在优先级关系，因为class的成员函数都是inline的，加了关键字inline也没什么特殊的。

【排序子序列】牛牛定义排序子序列为一个数组中一段连续的子序列,并且这段子序列是非递增或者非递减排序的。牛牛有一个长度为n的整数数组A,他现在有一个任 务是把数组A分为若干段排序子序列,牛牛想知道他最少可以把这个数组分为几段排序子序列.

如样例所示,牛牛可以把数组A划分为[1,2,3]和[2,2,1]两个排序子序列,至少需要划分为2个排序子序列,所以输出2

#include <iostream>

#include <vector>

using namespace std;

class Solution

{

public:

    int Fun(const vector<int> &nums)

    {

        int count = 0;

        int i = 0;

        while (i < nums.size())

        {

            if (nums[i] < nums[i + 1])

            {

                while (i < nums.size() && nums[i] < nums[i + 1])

                    ++i;

                ++count,++i;

                if (nums[i] == nums[i + 1])

                    ++i;

                if (nums[i] > nums[i + 1])

                {

                    while (i < nums.size() && nums[i] > nums[i + 1])

                        ++i;

                    ++count,++i;

                }

            }

        }

        return count;

    }

};

int main()

{

    Solution S;

    int n = 0;

    cin >> n;

    vector<int>nums(n,0);

    for (int i = 0;i < nums.size();++i)

        cin >> nums[i];

    int res = S.Fun(nums);

    cout<<res;

    return 0;

}

【倒置字符串】将一句话的单词进行倒置，标点不倒置。比如 I like beijing. 经过函数后变为：beijing. like I

//字符串倒置

#include <string>

#include <iostream>

using namespace std;

class Solution

{

public:

    string Fun( string &str)

    {

        if (str.empty())

            return "";

        reverse(str.begin(),str.end());

        auto it = str.begin();

        auto start = it;

        while (it != str.end())

        {

            while (it != str.end() && \*it != ' ')

                ++it;

            reverse(start,it);

            while (it != str.end() && \*it == ' ')

                ++it;

            start = it;

        }

        return str;

    }

};

int main()

{

    Solution S;

    string s;

    getline(cin,s);

    S.Fun(s);

    cout<<s<<endl;

    return 0;

}

day\_3

1. 以下能对二维数组a进行正确初始化的语句是（）

A int ta[2][]={{0,1,2},{3,4,5}};

B int ta[][3]={{0,1,2},{3,4,5}};

C int ta[2][4]={{0,1,2},{3,4},{5}};

D int ta[][3]={{0,,2},{},{3,4,5}};

解析:错选 A，正确选项：B，错误原因：二维数组的第二个参数永远不能省略，第一个参数可以

2.下面3段程序代码的效果一样吗？

 int b;

(1)const int \*a = &b;

(2)int const \*a = &b;

(3)int \*const a = &b;

A:(2)=(3); B:(1)=(3); C:(1)=(2); D:都不一样; E:都一样

解析:const在 \* 的左边，则指针指向的变量的值不可以直接通过指针改变(可以通过其他路径改变)；

         const在 \* 的右边，则指针的指向不可改变。简记为“左定值，右定向”。

         A:const int \*a = &b : const在 \* 的左边，则变量a的值不可以通过指针来改变

         B:int const \*a = &b : const在 \* 的左边，则变量a的值不可以通过指针来改变

         C:int \*const a = &b : const在 \* 的右边，则指针的指向不可以改变

         综上: (1) = (2)

3.关于vector.erase()的知识漏洞:

   1.vector<int>nuts{100,300,300,300,300,500};当删除第一个300的时候，会触发迭代器失效。

【字符串中找出连续最长的数字串】读入一个字符串str，输出字符串str中的连续最长的数字串

#include <iostream>

#include <string>

using namespace std;

class Solution

{

public:

    static bool IsNum(const char c)

    {

        return c >= '0' && c <= '9';

    }

public:

    string GetLongestNums(string &str,string &res)

    {

        int max = 0;//记录字符串的最大长度

        int count = 0;

        int end = 0;

        for (int i = 0;i < str.size();++i)

        {

            if (IsNum(str[i]))

            {

                ++count;

                if (count > max)

                {

                    end = i;

                    max = count;

                }

            }else{

                count = 0;

            }

        }

        res = str.substr(end - max + 1,max);

        return res;

    }

};

int main()

{

    string str,res;

    getline(cin,str);

    Solution S;

    res = S.GetLongestNums(str,res);

    cout<<res<<endl;

    return 0;

}

【数组中出现次数超过一半的数字】

int main()

{

    string str,res;

    getline(cin,str);

    Solution S;

    res = S.GetLongestNums(str,res);

    cout<<res<<endl;

    return 0;

}

class Solution {

public:

    int MoreThanHalfNum\_Solution(vector<int> numbers)

    {

        int x = 0,votes = 0;

        for (int num : numbers)

        {

            if (votes == 0)

                x = num;

            votes += num == x ? 1 : -1;

        }

        return x;

    }

};