CCF 编程能力等级认证
Grade Examination of Software Programming

C++ 四级

2024 年 12 月

1 单选题（每题 2 分，共 30 分）

第 1 题 下面的语句中，（ ）正确定义了一个计算浮点数x的平方($x^2 = x \times x$)的函数,并成功调用该函数。

A.

```
float square(float x) {
    return x * x;
}
float area = square(2);
```

B.

```
square(float x) {
    return x * x;
}
float area = square(2);
```

C.

```
void square(float x) {
    return x * x;
}
area = square(2.0);
```

D.

```
void square(float x) {
    x * x;
    return;
}
area = square(2);
```

第 2 题 下面代码的描述中，正确的是（ ）。

```
void n_chars(char c, int n) {
   while (n-- > 0)
        cout << c;
}

char my_char = 'w';
int times = 5;
n_chars(my_char, times);
```

- [ ] **A.** 代码执行结束后，`times` 的值为0
- [ ] **B.** `n` 是形参，`times` 是实参
- [ ] **C.** `n` 是实参，`times` 是形参
- [ ] **D.** 代码最后一行换成 `n_chars(times, my_char);` 也可以

第 3 题 给定以下代码，

```
void func(int& x) {
    x = x * 2;
}

int a = 5;
func(a);
```

执行上述代码后，变量 `a` 的值为（ ）。

- [ ] **A.** 5
- [ ] **B.** 10
- [ ] **C.** 15
- [ ] **D.** 20

第 4 题 运行下面代码，屏幕上输出是（ ）。

```
double* p_arr = new double [3];
p_arr[0] = 0.2;
p_arr[1] = 0.5;
p_arr[2] = 0.8;
p_arr += 1;
cout << p_arr[0] << endl;
p_arr -= 1;
delete p_arr;
```

- [ ] **A.** 0.2
- [ ] **B.** 0.5
- [ ] **C.** 1.2
- [ ] **D.** 1.5

第 5 题 运行下面代码片段后，`x` 和 `*p` 的结果分别是（ ）。

```
int x = 20;
int* p = &x;
*p = *p + 2;
```

- [ ] **A.** 20 20
- [ ] **B.** 20 22
- [ ] **C.** 22 20
- [ ] **D.** 22 22

第 6 题 下面的描述中，（ ）不能正确定义一个名为 `Student` 的结构体以及一个包含20个元素的结构数组。

- [ ] **A.**

```
struct Student {
    string name;
    int age;
    float score;
};
struct Student  students[20];
```

- [ ] **B.**

```
struct Student {
    string name;
    int age;
    float score;
};
Student  students[20];
```

- [ ] **C.**

```
struct Student {
    string name;
    int age;
    float score;
};
Student*  students = new Student[20];
```

- [ ] **D.**

```
struct Student {
    string name;
    int age;
    float score;
};
Student  students = new Student[20];
```

第 7 题 假定整型是32位，对一个2行3列的二维整数数组 `array` ,假设数组第一个元素在内存中的地址为 `0x7ffee4065820` ，则第2行第2个元素的地址 `&array[1][1]` 为（ ）。

```
int array[2][3] = {
    {0, 1, 2},
    {3, 4, 5}
};
```

- [ ] **A.** `0x7ffee4065824`
- [ ] **B.** `0x7ffee4065828`
- [ ] **C.** `0x7ffee406582c`
- [ ] **D.** `0x7ffee4065830`

第 8 题 下面（ ）正确定义二维数组。

- [ ] **A.** `int a[3][];`
- [ ] **B.** `int a[][];`
- [ ] **C.** `int a[][4];`
- [ ] **D.** `int a[][2] = {{1,2},{1,2},{3,4}};`

第 9 题 下面代码采用递推算法来计算斐波那契数列$f(n)=f(n-1)+f(n-2)$，则横线上应填写（ ）。

```
int fib(int n) {
    if (n == 0 || n == 1)
        return n;

    int f1 = 0;
    int f2 = 1;
    int result = 0;
    for (int i = 2; i <= n; i++) {
        ____________________________      // 在此处填入代码
    }
    return result;
}
```

- [ ] **A.**

```
result = f1 + f2;
f1 = f2;
f2 = result;
```

- [ ] **B.**

```
result += f1 + f2;
f1 = f2;
f2 = result;
```

- [ ] **C.**

```
result += f1 + f2;
f2 = result;
f1 = f2;
```

☐ **D.**

```
result = f1 + f2;
f2 = result;
f1 = f2;
```

第 10 题 下面关于排序算法（冒泡排序、插入排序和选择排序）的描述中，不正确的是（ ）。

☐ **A.** 冒泡排序基于元素交换实现，需借助临时变量，共涉及3个单元操作；而插入排序基于元素赋值实现，仅需1个单元操作。因此冒泡排序的计算开销通常比插入排序更高。

☐ **B.** 选择排序在任何情况下的时间复杂度都为$O\left(n^2\right)$。

☐ **C.** 冒泡排序在任何情况下的时间复杂度都为$O\left(n^2\right)$。

☐ **D.** 如果给定数据部分有序，插入排序通常比选择排序效率更高。

第 11 题 冒泡排序的第一轮操作是从左到右遍历数组，通过两两比较相邻元素，将当前最大的元素移动到末尾。给定数组 `arr[]={4, 1, 3, 1, 5, 2}` ，执行第一轮冒泡排序后数组 `arr` 中的内容为（ ）。

☐ **A.** `1, 4, 3, 1, 5, 2`

☐ **B.** `1, 3, 1, 4, 2, 5`

☐ **C.** `1, 4, 3, 1, 2, 5`

☐ **D.** `4, 1, 3, 1, 5, 2`

第 12 题 给定如下代码，其时间复杂度为（ ）。

```
int cellRecur(int n) {
    if (n == 1)
        return 1;
    return cellRecur(n - 1) + cellRecur(n - 1) + 1;
}
```

☐ **A.** $O(n^2)$

☐ **B.** $O(2^n)$

☐ **C.** $O(1)$

☐ **D.** $O(n)$

第 13 题 下面代码实现了插入排序函数，则横线上应填写（ ）。

```cpp
void insertion_sort(vector<int> &nums) {
    for (int i = 1; i < nums.size(); i++) {

        _____ {    // 在此处填入代码

        while (j >= 0 && nums[j] > base)

            nums[j + 1] = nums[j];
            j--;
        }
        nums[j + 1] = base;
    }
}
```

- [ ] **A.** `int base = nums[i], j = i - 1;`
- [ ] **B.** `int base = nums[i], j = i;`
- [ ] **C.** `int base = nums[0], j = i - 1;`
- [ ] **D.** `int base = nums[0], j = i;`

第 14 题 下面哪种方式不能实现将字符串"Welcome to GESP!"输出重定向到文件 `log.txt` （ ）。

- [ ] **A.**

```cpp
freopen("log.txt", "w", stdout);
cout << "Welcome to GESP!" << endl;
fclose(stdout);
```

- [ ] **B.**

```cpp
std::ofstream outFile("log.txt");
outFile << "Welcome to GESP!" << endl;
outFile.close();
```

- [ ] **C.**

```cpp
std::ofstream outFile("log.txt");
cout << "Welcome to GESP!" << endl;
outFile.close();
```

- [ ] **D.**

```cpp
ofstream log_file("log.txt");
streambuf* org_cout = cout.rdbuf();
cout.rdbuf(log_file.rdbuf());
cout << "This output will go to the log file." << endl;
cout.rdbuf(oorg_cout);
```

第 15 题 运行下面的代码，将出现什么情况？ （ ）

```cpp
double hmean(double a, double b) {
    if (a == -b )
        throw runtime_error("Runtime error occurred");
```

```
    return 2.0*a*b/(a + b);
}

int main() {
    double x = 10;
    double y = -10;

    try {
        int result = hmean(x, y);
        cout << "hmean: " << result << endl;
    }
    catch (const runtime_error& e) {
        cout << "Caught: " << e.what() << endl;
    } catch (...) {
        cout << "Caught an unknown exception." << endl;
    }
    return 0;
}
```

- [ ] **A.** 屏幕上输出 `Caught: Runtime error occurred`
- [ ] **B.** 屏幕上输出 `Caught an unknown exception`
- [ ] **C.** 程序调用 std::terminate()
- [ ] **D.** 编译错误

2 判断题（每题 2 分，共 20 分）

第 1 题 在 C++ 中，下面代码可以正确定义指针和初始化指针。

```
int* ptr;
*ptr = 10;
```

第 2 题 一个函数必须在调用之前既声明又定义。

第 3 题 函数参数可以通过值传递、引用传递和指针传递，这样函数内对参数的修改可以直接修改传入变量的值。

第 4 题 `int arr[3][]` 是一个正确的二维数组的声明。

第 5 题 递推是一种通过已知的初始值和递推公式，逐步求解目标值的算法。

第 6 题 某算法的递推关系式为 $T(n) = T(n-1) + n$（n为正整数）及 $T(0) = 1$，则该算法的时间复杂度为$O(n^2)$。

第 7 题 冒泡排序的平均时间复杂度为 $O(n^2)$，但最优情况下为$O(n)$。

第 8 题 冒泡排序和插入排序都是稳定的排序算法。

第 9 题 选择排序是稳定的排序算法。

第 10 题 在 C++语言中，如果一个函数可能抛出异常，那么一定要在try 子句里调用这个函数。

3　编程题（每题 25 分，共 50 分）

3.1　编程题 1

- **试题名称**：Recamán
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.1.1　题目描述

小杨最近发现了有趣的 Recamán 数列，这个数列是这样生成的：

- 数列的第一项 a_1 是 1；
- 如果 $a_{k-1}-k$ 是正整数并且没有在数列中出现过，那么数列的第 k 项 a_k 为 $a_{k-1}-k$，否则为 $a_{k-1}+k$。

小杨想知道 Recamán 数列的前 n 项从小到大排序后的结果。手动计算非常困难，小杨希望你能帮他解决这个问题。

3.1.2　输入格式

第一行，一个正整数 n。

3.1.3　输出格式

一行，n 个空格分隔的整数，表示 Recamán 数列的前 n 项从小到大排序后的结果。

3.1.4　样例

3.1.5　输入样例 1

```
5
```

3.1.6　输出样例 1

```
1 2 3 6 7
```

3.1.7　输入样例 2

```
8
```

3.1.8　输出样例 2

```
1 2 3 6 7 12 13 20
```

3.1.9　样例解释

对于样例 1，$n=5$：

- $a_1=1$；
- $a_1-2=-1$，不是正整数，因此 $a_2=a_1+2=3$；
- $a_2-3=0$，不是正整数，因此 $a_3=a_2+3=6$；
- $a_3-4=2$，是正整数，且没有在数列中出现过，因此 $a_4=2$；

- $a_4 - 5 = -3$，不是正整数，因此 $a_5 = a_4 + 5 = 7$；

a_1, a_2, a_3, a_4, a_5 从小到大排序后的结果为 `1 2 3 6 7`。

3.1.10 数据范围

对于所有数据点，保证 $1 \leq n \leq 3000$。

3.2 编程题 2

- **试题名称**：字符排序
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.2.1 题面描述

小杨有 n 个仅包含小写字母的字符串 $s_1, s_2, \ldots, s_n$，小杨想将这些字符串按一定顺序排列后拼接到一起构成字符串 t 。小杨希望最后构成的字符串 t 满足：

- 假设 t_i 为字符串 t 的第 i 个字符，对于所有的 $j < i$ 均有 $t_j \leq t_i$。两个字符的大小关系与其在字母表中的顺序一致，例如 $e < g < p < s$。

小杨想知道是否存在满足条件的字符串排列顺序。

3.2.2 输入格式

第一行包含一个正整数 T，代表测试数据组数。

对于每组测试数据，第一行包含一个正整数 n，含义如题面所示。

之后 n 行，每行包含一个字符串 s_i。

3.2.3 输出格式

对于每组测试数据，如果存在满足条件的排列顺序，输出 1，否则输出 0。

3.2.4 样例

```
3
3
aa
ac
de
2
aac
bc
1
gesp
```

```
1
0
0
```

3.2.5 样例解释

对于第一组测试数据，一种可行的排列顺序为 aa+ac+de，构成的字符串 t 为 aaacde，满足条件。

对于全部数据，保证有 $1 \leq t, n \leq 100$，每个字符串的长度不超过 10。