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

C++ 六级

2024 年 06 月

1 单选题（每题 2 分，共 30 分）

第 1 题 面向对象的编程思想主要包括（ ）原则。

- ☐ **A.** 贪心、动态规划、回溯
- ☐ **B.** 并发、并行、异步
- ☐ **C.** 递归、循环、分治
- ☐ **D.** 封装、继承、多态

第 2 题 运行下列代码，屏幕上输出（ ）。

```
#include <iostream>
using namespace std;

class my_class {
public:
    static int count;
    my_class() {
        count++;
    }
    ~my_class() {
        count--;
    }
    static void print_count() {
       cout  << count << " ";
    }
};
int my_class::count = 0;
int main() {
    my_class obj1;
    my_class::print_count();
    my_class obj2;
    obj2.print_count();
    my_class obj3;
    obj3.print_count();
    return 0;
}
```

- ☐ **A.** 1 1 1

☐ **B.** 1 2 3

☐ **C.** 1 1 2

☐ **D.** 1 2 2

第 3 题 运行下列代码，屏幕上输出（ ）。

```
#include <iostream>
using namespace std;

class shape {
protected:
    int width, height;
public:
    shape(int a = 0, int b = 0) {
        width = a;
        height = b;
    }
    virtual int area() {
        cout << "parent class area: " <<endl;
        return 0;
    }
};

class rectangle: public shape {
public:
    rectangle(int a = 0, int b = 0) : shape(a, b) { }

    int area () {
        cout << "rectangle area: ";
        return (width * height);
    }
};

class triangle: public shape {
public:
    triangle(int a = 0, int b = 0) : shape(a, b) { }

    int area () {
        cout << "triangle area: ";
        return (width * height / 2);
    }
};

int main() {
   shape *pshape;
   rectangle rec(10, 7);
   triangle  tri(10, 5);

   pshape = &rec;
   pshape->area();

   pshape = &tri;
   pshape->area();
   return 0;
```

```
}
```

- [ ] **A.** rectangle area: triangle area:
- [ ] **B.** parent class area: parent class area:
- [ ] **C.** 运行时报错
- [ ] **D.** 编译时报错

第 4 题 向一个栈顶为hs的链式栈中插入一个指针为s的结点时，应执行（ ）。

- [ ] **A.** `hs->next = s;`
- [ ] **B.** `s->next = hs; hs = s;`
- [ ] **C.** `s->next = hs->next; hs->next = s;`
- [ ] **D.** `s->next = hs; hs = hs->next;`

第 5 题 在栈数据结构中，元素的添加和删除是按照什么原则进行的?

- [ ] **A.** 先进先出
- [ ] **B.** 先进后出
- [ ] **C.** 最小值先出
- [ ] **D.** 随机顺序

第 6 题 要实现将一个输入的十进制正整数转化为二进制表示，下面横线上应填入的代码为（ ）。

```
#include <iostream>
using namespace std;

stack<int> ten2bin(int n) {
    stack<int> st;
    int r, m;

    r = n % 2;
    m = n / 2;
    st.push(r);

    while (m != 1) {
        r = m % 2;
        st.push(r);
        m = m / 2;
    }
    st.push(m);
    return st;
}

int main() {
    int n;
    cin >> n;
    stack<int> bin;
    bin = ten2bin(n);
    while (!bin.empty()) {
```

```
        ____________________ // 在此处填入代码
    }
    return 0;
}
```

- [ ] **A.** `cout << bin.top();  bin.pop();`
- [ ] **B.** `bin.pop(); cout << bin.top();`
- [ ] **C.** `cout << bin.back();  bin.pop();`
- [ ] **D.** `cout << bin.front();  bin.pop();`

第 7 题 下面定义了一个循环队列的类，请补全判断队列是否满的函数，横向上应填写（ ）。

```
#include <iostream>

using namespace std;

class circular_queue {
private:
    int *arr; // 数组用于存储队列元素
    int capacity; // 队列容量
    int front; // 队头指针
    int rear; // 队尾指针

public:
    circular_queue(int size) {
        capacity = size + 1; // 为了避免队列满时与队列空时指针相等的情况，多预留一个空间
        arr = new int[capacity];
        front = 0;
        rear = 0;
    }

    ~circular_queue() {
        delete[] arr;
    }

    bool is_empty() {
        return front == rear;
    }

    bool is_full() {
        _________________ // 在此处填入代码
    }

    void en_queue(int data) {
        if (is_full()) {
            cout << "队列已满，无法入队！" << endl;
            return -1;
        }
        arr[rear] = data;
        rear = (rear + 1) % capacity;
        return 1;
    }

```

```
    int de_queue() {
        if (is_empty()) {
            cout << "队列为空，无法出队！" << endl;
            return -1; // 出队失败，返回一个特殊值
        }
        int data = arr[front];
        front = (front + 1) % capacity;
        return data;
    }
};
```

- [ ] **A.** `return (rear + 1) % capacity == front;`
- [ ] **B.** `return rear % capacity == front;`
- [ ] **C.** `return rear == front;`
- [ ] **D.** `return (rear + 1) == front;`

第 8 题 对“classmycls”使用哈夫曼（Huffman）编码，最少需要（ ）比特。

- [ ] **A.** 10
- [ ] **B.** 20
- [ ] **C.** 25
- [ ] **D.** 30

第 9 题 二叉树的（ ）第一个访问的节点是根节点。

- [ ] **A.** 先序遍历
- [ ] **B.** 中序遍历
- [ ] **C.** 后序遍历
- [ ] **D.** 以上都是

第 10 题 一棵5层的满二叉树中节点数为（ ）。

- [ ] **A.** 31
- [ ] **B.** 32
- [ ] **C.** 33
- [ ] **D.** 16

第 11 题 在求解最优化问题时，动态规划常常涉及到两个重要性质，即最优子结构和( )。

- [ ] **A.**

  重叠子问题
- [ ] **B.** 分治法
- [ ] **C.** 贪心策略
- [ ] **D.** 回溯算法

第 12 题 青蛙每次能跳1或2步，下面代码计算青蛙跳到第n步台阶有多少种不同跳法。则下列说法，错误的是( )。

```
int jump_recur(int n) {
    if (n == 1) return 1;
    if (n == 2) return 2;
    return jump_recur(n - 1) + jump_recur(n - 2);
}

int jump_dp(int n) {
    vector<int> dp(n + 1); // 创建一个动态规划数组，用于保存已计算的值
    // 初始化前两个数
    dp[1] = 1;
    dp[2] = 2;
    // 从第三个数开始计算斐波那契数列
    for (int i = 3; i <= n; ++i) {
        dp[i] = dp[i - 1] + dp[i - 2];
    }
    return dp[n];
}
```

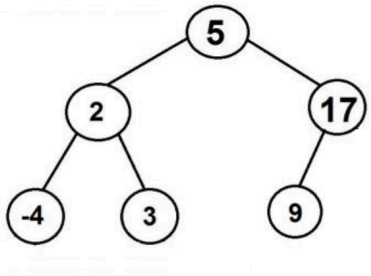
- ☐ **A.** 函数jump_recur()采用递归方式。
- ☐ **B.** 函数jump_dp()采用动态规划方法。
- ☐ **C.** 当n较大时，函数jump_recur()存在大量重复计算，执行效率低。
- ☐ **D.** 函数jump_recur()代码量小，执行效率高。

第 13 题 阅读以下二叉树的广度优先搜索代码:

```
#include <iostream>
#include <queue>

using namespace std;

// 二叉树节点的定义
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

// 宽度优先搜索（BFS）迭代实现
TreeNode* bfs(TreeNode* root, int a) {
    if (root == nullptr) return nullptr;

    queue<TreeNode*> q;
    q.push(root);

    while (!q.empty()) {
        TreeNode* node = q.front();
        q.pop();

        if (node->val == a)
            return node;
```

```
27
28          cout << node->val << " "; // 先访问当前节点
29
30          if (node->left) q.push(node->left); // 将左子节点入队
31          if (node->right) q.push(node->right); // 将右子节点入队
32      }
33      return nullptr;
34  }
```

使用以上算法，在以下这棵树搜索数值20时，可能的输出是( )。



- ☐ **A.** 5 2 -4 3 17 9
- ☐ **B.** -4 2 3 5 9 17
- ☐ **C.** 5 2 17 -4 3 9
- ☐ **D.** 以上都不对

第 14 题 同上题中的二叉树，阅读以下二叉树的深度优先搜索代码:

```
1   #include <iostream>
2   #include <stack>
3
4   using namespace std;
5
6   // 非递归深度优先搜索（DFS）
7   TreeNode* dfs(TreeNode* root, int a) {
8       if (root == nullptr) return nullptr;
9
10      stack<TreeNode*> stk;
11      stk.push(root);
12
13      while (!stk.empty()) {
14          TreeNode* node = stk.top();
15          stk.pop();
16          if (node->val == a)
17              return node;
18
19          cout << node->val << " "; // 访问当前节点
20
21          if (node->right) stk.push(node->right); // 先压入右子节点
22          if (node->left) stk.push(node->left); // 再压入左子节点
23      }
24      return nullptr;
25  }
```

使用以上算法，在二叉树搜索数值20时，可能的输出是( )。

- ☐ **A.** 5 2 -4 3 17 9
- ☐ **B.** -4 2 3 5 9 17
- ☐ **C.** 5 2 17 -4 3 9
- ☐ **D.** 以上都不对

第 15 题 在上题的树中搜索数值3时，采用深度优先搜索一共比较的节点数为（ ）。

- ☐ **A.** 2
- ☐ **B.** 3
- ☐ **C.** 4
- ☐ **D.** 5

2　判断题（每题 2 分，共 20 分）

第 1 题 哈夫曼编码本质上是一种贪心策略。

第 2 题 创建一个对象时，会自动调用该对象所属类的构造函数。如果没有定义构造函数，编译器会自动生成一个默认的构造函数。

第 3 题 定义一个类时，必须手动定义一个析构函数，用于释放对象所占用的资源。

第 4 题 C++中类内部可以嵌套定义类。

第 5 题 000, 001, 011, 010, 110, 111, 101, 100是一组格雷码。

第 6 题 n个节点的双向循环链表，在其中查找某个节点的平均时间复杂度是$O(\log n)$。

第 7 题 完全二叉树可以用数组存储数据。

第 8 题 在C++中，静态成员函数只能访问静态成员变量。

第 9 题 在深度优先搜索中，通常使用队列来辅助实现。

第 10 题 对0-1背包问题，贪心算法一定能获得最优解。