GESP一级考点分析

级别	知识内容（C++）
一级	计算机基础与编程环境 计算机历史 变量的定义与使用 基本数据类型（整型、浮点型、字符型、布尔型） 控制语句结构（顺序、循环、选择） 基本运算（算术运算、关系运算、逻辑运算） 输入输出语句

1.计算机历史

> 详细参考信息学发展史(近代版)

- 艾伦·图灵：提出图灵机概念，奠定现代计算机科学理论基础
- 克劳·德香农：提出了信息论的基本框架。香农提出了“比特”（bit）作为信息的基本度量单位，并定义了信息熵，描述了信息的不确定性。
- 冯·诺依曼架构：设计了一个具有存储程序能力的计算机架构，这一架构被称为“冯·诺依曼架构”。它将计算机的五个基本组件：输入设备、输出设备、存储器、控制单元和算术逻辑单元（ALU）分离开来。这个架构成为现代计算机设计的基础。
- **ENIAC**：1946年，ENIAC（电子数值积分计算机）是世界上第一台通用电子计算机。

2.变量

- **定义**：在 C++ 中，定义变量时会分配内存空间。例如：

```
int age; // 声明一个整数变量 age
```

- **声明**：可以在同一行中同时声明和定义多个变量：

```
nt x = 5, y = 10; // 同时声明并初始化两个整数变量
```

3.数据类型

C++ 提供了一系列内置数据类型，用于存储不同类型的数据。

整数类型

- **int**：标准整数类型，通常为 4 字节（32 位），表示的范围一般为 -2,147,483,648 到 2,147,483,647。
- **short**：短整型，通常为 2 字节（16 位），范围为 -32,768 到 32,767。
- **long**：长整型，通常为 4 字节（32 位）或 8 字节（64 位），取决于平台。
- **long long**：至少为 8 字节（64 位），用于表示更大的整数。
- **unsigned**：无符号整数类型，不包含负值。例如，`unsigned int` 的范围为 0 到 4,294,967,295。

浮点数类型

- **float**：单精度浮点数，通常为 4 字节，适合存储较小范围的浮点数。精度约为 6 位有效数字。
- **double**：双精度浮点数，通常为 8 字节，适合存储更大范围的浮点数。精度约为 15 位有效数字。
- **long double**：长双精度浮点数，通常为 10、12 或 16 字节，具体取决于实现，提供更高的精度。

字符类型

- **char**：用于存储单个字符，通常为 1 字节（8 位）。可以表示 ASCII 字符集中的字符。
- **wchar_t**：宽字符类型，通常为 2 或 4 字节，主要用于表示 Unicode 字符。

布尔类型

- **bool**：表示布尔值，只有两个取值：`true`（真）和 `false`（假）。通常为 1 字节。

4.运算符

C++ 中的运算符用于执行各种操作，分为几类。以下是运算符的详细介绍：

1. 算术运算符

这些运算符用于执行数学计算。

运算符	描述	示例
`+`	加法	`a + b`
`-`	减法	`a - b`
`*`	乘法	`a * b`
`/`	除法（整除和浮点数）	`a / b`
`%`	取模（余数）	`a % b`

2. 关系运算符

这些运算符用于比较两个值，并返回布尔值 (`true` 或 `false`)。

运算符	描述	示例
`==`	等于	`a == b`
`!=`	不等于	`a != b`
`>`	大于	`a > b`
`<`	小于	`a < b`
`>=`	大于或等于	`a >= b`
`<=`	小于或等于	`a <= b`

3. 逻辑运算符

用于执行布尔逻辑运算。

运算符	描述	示例
`&&`	逻辑与（AND）	`a && b`
`||`	逻辑或（OR）	`a || b`
`!`	逻辑非（NOT）	`!a`

5. 赋值运算符

用于给变量赋值。可以与其他运算符结合使用。

运算符	描述	示例
`=`	赋值	`a = b`
`+=`	加法赋值	`a += b`

运算符	描述	示例
`-=`	减法赋值	`a -= b`
`*=`	乘法赋值	`a *= b`
`/=`	除法赋值	`a /= b`
`%=`	取模赋值	`a %= b`
`&=`	按位与赋值	`a &= b`
`|=`	按位或赋值	`a |= b`
`^=`	按位异或赋值	`a ^= b`
`<<=`	左移赋值	`a <<= 2`
`>>=`	右移赋值	`a >>= 2`

6. 自增和自减运算符

用于增加或减少变量的值。

运算符	描述	示例
`++`	自增（++a 或 a++）	`a++`（后置），`++a`（前置）
`--`	自减（--a 或 a--）	`a--`（后置），`--a`（前置）

5.逻辑结构

在 C++ 编程中，逻辑结构主要涉及控制程序流的结构。逻辑结构决定了代码执行的顺序以及条件和循环的处理方式。以下是 C++ 中的主要逻辑结构：

1. 顺序结构

顺序结构是指程序从上到下逐行执行，没有任何跳转或分支。最基本的代码执行方式。

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

int main() {
    cout << "Hello, World!" << endl;
    cout << "This is a sequential structure." << endl;
    return 0;
}
```

2. 选择结构（条件结构）

选择结构允许程序根据条件执行不同的代码块。C++ 中常用的选择结构有 `if`、`else if` 和 `switch`。

2.1 `if` 语句

```
if (condition) {
    // 条件为真时执行的代码
} else {
    // 条件为假时执行的代码
}
```

示例：

```
int a = 10;
if (a > 0) {
    cout << "a is positive." << endl;
} else {
    cout << "a is non-positive." << endl;
}
```

2.2 `else if` 语句

可以处理多个条件。

```
if (condition1) {
    // 条件1为真时执行的代码
} else if (condition2) {
    // 条件1为假 条件2为真时执行的代码
} else {
    // 所有条件都为假时执行的代码
}
```

示例：

```
int score = 85;
if (score >= 90) {
    cout << "Grade: A" << endl;
} else if (score >= 80) {
    cout << "Grade: B" << endl;
} else {
    cout << "Grade: C" << endl;
}
```

2.3 `switch` 语句

用于基于变量的值执行不同的代码块。

```
switch (expression) {
    case constant1:
        // 执行代码块1
        break;
    case constant2:
        // 执行代码块2
        break;
    default:
        // 默认执行的代码块
}
```

示例：

```
int day = 3;
switch (day) {
    case 1:
        cout << "Monday" << endl;
        break;
    case 2:
        cout << "Tuesday" << endl;
        break;
    case 3:
        cout << "Wednesday" << endl;
        break;
    default:
        cout << "Invalid day" << endl;
}
```

3. 循环结构

循环结构允许重复执行代码块，直到满足特定条件。C++ 中常用的循环结构有 `for`、`while` 和 `do...while`。

3.1 `for` 循环

用于在已知次数的情况下执行循环。

```
for (initialization; condition; increment) {
    // 循环体
}
```

示例：

```
for (int i = 0; i < 5; i++) {
    cout << "Iteration: " << i << endl;
}
```

3.2 `while` 循环

在条件为真时重复执行代码块。

```
while (condition) {
    // 循环体
}
```

示例：

```cpp
int i = 0;
while (i < 5) {
    cout << "Iteration: " << i << endl;
    i++;
}
```

3.3 `do...while` 循环

先执行一次循环体，然后检查条件。

```cpp
do {
    // 循环体
} while (condition);
```

示例：

```cpp
int i = 0;
do {
    cout << "Iteration: " << i << endl;
    i++;
} while (i < 5);
```

4. 跳转结构

跳转结构用于改变程序的执行流，常见的有 `break`、`continue` 和 `return`。

- `break`：终止当前循环或 `switch` 语句。
- `continue`：跳过当前循环的剩余部分，开始下一次循环。
- `return`：从函数返回，结束函数的执行。

示例：

```cpp
for (int i = 0; i < 10; i++) {
    if (i == 5) {
        break; // 退出循环
    }
    cout << "Iteration: " << i << endl;
}

for (int i = 0; i < 10; i++) {
    if (i % 2 == 0) {
        continue; // 跳过偶数
    }
    cout << "Odd number: " << i << endl;
}
```

6.输入输出语句

在 C++ 中，输入输出操作主要通过流（stream）来实现。以下是关于输入输出语句的详细介绍：

1. 输出语句

C++ 使用 `cout` 对象进行输出，通常配合流插入运算符 `<<` 使用。

示例：

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

int main() {
    int age = 20;
    cout << "Your age is: " << age << endl;  // 输出：Your age is: 20
    return 0;
}
```

格式化输出

C++ 可以通过 `<iomanip>` 头文件中的函数进行格式化输出。

- `setw(int n)`：设置输出字段宽度。
- `setprecision(int n)`：设置浮点数的精度。

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

int main() {
    double pi = 3.14159;
    cout << fixed << setprecision(2) << "Pi: " << pi << endl;  // 输出：Pi: 3.14
    cout << setw(10) << "Hello" << endl;  // 输出：       Hello
    return 0;
}
```

2. 输入语句

C++ 使用 `cin` 对象进行输入，通常配合流提取运算符 `>>` 使用。

示例：

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

int main() {
    int age;
    cout << "Enter your age: ";
    cin >> age;  // 输入：20
    cout << "Your age is: " << age << endl;  // 输出：Your age is: 20
    return 0;
}
```

处理输入错误

可以通过检查 `cin` 的状态来处理输入错误。

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

int main() {
    int age;
    cout << "Enter your age: ";
    if (cin >> age) {
        cout << "Your age is: " << age << endl;
    } else {
        cout << "Invalid input!" << endl;
    }
    return 0;
}
```

读取多个输入

可以使用空格或换行符分隔多个输入。

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

int main() {
    int a, b;
    cout << "Enter two numbers: ";
    cin >> a >> b;  // 输入：5 10
    cout << "Sum: " << (a + b) << endl;  // 输出：Sum: 15
    return 0;
}
```