第3章

程序基本控制结构

结构化程序设计的基础包括顺序结构、选择结构和循环结构。之前涉及的例题都使用了顺序结构完成编程任务。然而，在实际应用系统中，常常需要根据不同情况执行不同处理流程。举例来说，电子商务网站可能会免除一部分用户的运费，这就需要根据购买金额进行选择；保险公司需要根据投保人的不同条件采用不同算法计算保费；杀毒软件需要重复扫描磁盘文件并判断其是否感染病毒；搜索引擎需要反复查找索引表以满足用户的查询要求；等等。以上这些功能都需要通过选择与循环控制结构实现。

本章将带领读者深入了解结构化程序设计，并介绍基本的编程结构，包括顺序结构、选择结构和循环结构。

3.1 逻辑运算符和逻辑表达式

逻辑运算符一般有 3 个，分别是“&&”(与)、“||”(或)、“!”(非)。

- “与”运算符：使用符号“&&”表示，表示当两个条件同时为真时，整个条件才是真的。
- “或”运算符：使用符号“||”表示，表示当两个条件中至少有一个为真时，整个条件就是真的。
- “非”运算符：使用符号“!”表示，经常用于对一个条件取反，如果原本为真，则取反后就变成假的；如果原本为假，则取反后就变成真的。

下面以例 3.1 为例讲解使用 Copilot 求解逻辑表达式的运算，令其根据已给出的代码进行运算，并给出结果。

【例 3.1】 引导 Copilot 分析代码中的逻辑表达式并进行运算，最后返回结果。

利用 Copilot 进行运算的结果如下：

```
#include <stdio.h>

int main()
{
    int a=1,b=10,c=1,x,y;
    x=a && b || ++c;
```

```
    printf("x=%d,c=%d\n",x,c);
    y=!a && --b || ++c;
    printf("y=%d,b=%d,c=%d\n",y,b,c);
    return 0;
}
//请给出以上代码的运算结果
//将答案输出在下方
//x=1,c=2
//y=0,b=9,c=3
```

3.2 选择控制结构

3.2.1 if 单分支语句

if 单分支语句是一种常见的条件控制语句,用于在满足特定条件时执行某个代码块,否则程序将跳过此代码块并继续执行下一行代码。其基本语法格式如下:

```
if (condition) {
//condition 为真时执行的代码块
}
```

其中,condition 表示要测试的条件表达式,可以是任何返回布尔值(true 或 false)的表达式,如果该条件为真,则执行花括号中的代码块;如果条件为假,则直接跳过该代码块。


例 3.2
视频讲解

【例 3.2】 从键盘输入两个整数,分别将值存入 a 和 b,并判断 a 和 b 的大小,若 a 大于 b,则交换双方的值再输出,否则直接输出。

引导 Copilot 生成代码的自然语言如下:

```
//输入两个整数存到 a 和 b 中并比较大小,a>b 交换 a 和 b 再输出,否则直接输出
```

其生成的代码如下:

```
#include <stdio.h>
int main()
{
    int a,b;
    printf("请输入两个整数:");
    scanf("%d%d",&a,&b);
    if(a>b)
{
    int t;
    t=a;
    a=b;
    b=t;
}
    printf("a=%d,b=%d",a,b);
```

```
    return 0;
}
```

3.2.2　if 双分支语句

if 双分支语句是一种条件控制语句，用于在满足特定条件时执行一个代码块，否则执行另一个代码块。基本的 if 双分支语句的语法格式如下：

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

其中，condition 表示要测试的条件表达式，如果该条件为真，则执行 if 语句后面花括号中的代码块；如果条件为假，则执行 else 语句后面花括号中的代码块。因此，又称之为 if-else 语句。

【例 3.3】　编写程序，判断一个年份是否为闰年。

引导 Copilot 生成代码的自然语言如下：

```
//请编写一个程序，其功能是判断一个年份是否为闰年
```

其生成的代码如下：

```
#include  <stdio.h>

int main()
{
    int year;
    printf("Please input a year:");
    scanf("%d",&year);
    if(year%4==0&&year%100!=0||year%400==0)
    printf("%d is a leap year.",year);
    else
    printf("%d is not a leap year.",year);
    return 0;
}
```

3.3　条件表达式

在 C 语言中，条件表达式是一种返回布尔值(true 或 false)的表达式，它通常与 if 语句、while 循环、for 循环等条件控制语句配合使用，用于根据给定的条件判断程序的执行路径。本节只针对条件表达式的使用方法与格式做出解释与讲解。

条件表达式的基本语法格式如下：

```
condition ?(value_if_true) : (value_if_false)
```

其中,condition 表示要测试的条件表达式,(value_if_true)表示当 condition 为真时返回的值,(value_if_false)表示当 condition 为假时返回的值。如果 condition 为真,则返回(value_if_true),否则返回(value_if_false)。

例如,在 if 语句中使用条件表达式:

```
int a = 10, b = 20;
if (a > b) {
    printf("a is greater than b");
} else {
    printf("b is greater than a");
}
```

可以使用条件表达式简化如下:

```
int a = 10, b = 20;
printf(a > b ?"a is greater than b" : "b is greater than a");
```

这样代码更简洁,具有更好的可读性和可维护性。

3.4 switch 多分支语句

在 C 语言中,switch 多分支语句是一种根据不同情况执行不同代码块的流程控制语句,它通常与 case 标签、default 标签等语法结构配合使用,用于根据给定的条件判断程序的执行路径。

switch 多分支语句的基本语法格式如下:

```
switch (expression) {
    case constant1:
        //常量表达式为 constant1 时执行的代码块
        break;
    case constant2:
        //常量表达式为 constant2 时执行的代码块
        break;
    ...
    default:
        //其他情况下执行的代码块(可选)
        break;
}
```

其中,expression 表示要测试的表达式,case 后跟的 constant 表示要匹配的常量值。当 expression 和某个 case 后面的 constant 相等时,就执行对应的代码块。如果匹配不到对应的 case,则执行 default 标签后的代码块(如果有)。

例如:

```
int n = 3;
switch (n) {
    case 1:
        printf("Monday");
        break;
    case 2:
        printf("Tuesday");
        break;
    case 3:
        printf("Wednesday");
        break;
    default:
        printf("Other day");
        break;
}
```

以上代码将输出 Wednesday，因为 n 的值为 3，所以匹配了第 3 个 case 标签。

需要注意的是，每个 case 代码块中必须包含一个 break 语句，用于退出 switch 语句，否则会导致后面的 case 代码块也被执行，直到遇到 break 或结束整个 switch 语句为止。这也是很多初学者容易犯的错误之一。

【例 3.4】 用程序实现从键盘输入两个浮点数并存入变量之后，显示一个菜单并根据用户选择的选项分别执行加、减、乘、除四则运算。

引导 Copilot 生成代码的自然语言如下：

```
//请写一段代码实现一个简单的计算器,要求如下:
//1.根据用户输入的数字选择要进行的操作,1 加法,2 减法,3 乘法,4 除法
//2.根据用户输入的数字进行相应的计算,计算结果保留两位小数
```

Copilot 生成的代码如下：

```
#include <stdio.h>
int main()
{
    int a,b,c;
    printf("请输入一个数字选择计算方式:1 加法,2 减法,3 乘法,4 除法");
    scanf("%d",&a);
    printf("please input two numbers:");
    scanf("%d%d",&b,&c);
    switch(a)
    {
        case 1:printf("%d+%d=%d",b,c,b+c);break;
        case 2:printf("%d-%d=%d",b,c,b-c);break;
        case 3:printf("%d*%d=%d",b,c,b*c);break;
        case 4:printf("%d/%d=%d",b,c,b/c);break;
        default:printf("error");
    }
    return 0;
}
```

3.5 循环控制语句

3.5.1 while循环语句

在C语言中,while语句是一种基本的迭代控制结构,用于循环执行一段语句,前提是给定的条件为真(true)。

while语句的基本语法格式如下:

```
while (condition) {
    //循环体代码块
}
```

其中,condition表示要测试的条件表达式,当它的值为真时,将重复执行花括号内的代码块;如果condition的值为假,则跳过while循环并继续执行下面的代码。

以下代码演示了一个简单的while循环:

```
int i = 0;
while (i < 5){
    printf("%d\n", i);
    i++;
}
```

以上代码将输出数字0～4,循环体中的代码将执行5次,直到i的值增加到5时,条件变为假,循环结束。

需要注意的是,在while循环中应谨慎处理循环条件,以防无限循环或者跳出循环之前没有满足预期条件的问题。另外,为了避免无限循环,while循环体内应包含使循环条件可以转化成假的操作或条件判断,例如改变计数器变量等。

【例3.5】 编写代码,计算从1加到100。

引导Copilot生成代码的自然语言如下:

```
//我需要计算整数1到100的和
```

Copilot生成的代码如下:

```
#include <stdio.h>

int main()
{
    int i,sum=0;
    for(i=1;i<=100;i++)
        sum+=i;
    printf("The sum of 1 to 100 is %d.",sum);
    return 0;
}
```

3.5.2　for 循环语句

在 C 语言中，for 语句是一种基本的迭代控制结构，用于循环执行一段代码块，可以更方便地控制迭代次数和步长。

for 循环的基本语法格式如下：

```
for (initialization; condition; update) {
    //循环体代码块
}
```

其中，initialization 表示循环开始前执行的初始化表达式，通常用于定义计数器变量及其初始值；condition 表示要测试的条件表达式，当它的值为真(true)时，将重复执行循环体内的代码块；update 表示每次迭代之后要执行的语句，通常用于更新计数器变量或者其他状态变量。

以下代码演示了一个简单的 for 循环：

```
for (int i = 0; i < 5; i++) {
    printf("%d\n", i);
}
```

以上代码将输出数字 0～4，循环体中的代码将执行 5 次，直到 i 的值增加到 5 时，条件变为假，循环结束。

需要注意的是，在 for 循环中，可以省略 initialization、condition、update 中的任意一项，但分号必须保留。如果没有提供 condition，则默认循环条件为真。如果没有提供 update，则循环体内必须包含修改计数器变量的语句，以便在循环中停止。

【例 3.6】　计算 1/(1＋5i)(0＜i＜101)中 i 从 1 到 100 的和。

引导 Copilot 生成代码的自然语言如下：

```
//我需要一个程序计算 1/(1+5i)(0<i<101)中 i 从 1 到 100 的和
```


例 3.6
视频讲解

Copilot 生成的代码如下：

```
#include <stdio.h>

int main()
{
    int i;
    double sum=0;
    for(i=1;i<=100;i++)
    {
        sum=sum+1.0/(1+5*i);
    }
    printf("The sum of i=1 to 100 is %lf",sum);
    return 0;
}
```

3.5.3 do-while 循环语句

在 C 语言中,do-while 语句也是一种迭代控制结构,它与 while 语句类似,不同之处在于其循环体至少执行一次,这是因为条件检查在循环体的末尾才进行。

do-while 循环的基本语法格式如下:

```
do {
    //循环体代码块
} while (condition);
```

其中,condition 表示循环条件,通过该条件判断是否继续循环。在每次执行完循环体后都会检查 condition 的值,只有在 condition 为真时才继续循环,否则跳出循环。

以下代码演示了一个简单的 do-while 循环:

```
int i = 0;
do {
    printf("%d\n", i);
    i++;
} while (i < 5);
```

以上代码将输出数字 0～4,即使 i 的初始值为 0,do-while 循环体仍然会执行一次,直到 i 的值增加到 5 时,条件变为假,循环结束。

需要注意的是,在使用 do-while 循环时,应注意循环条件和循环体内的语句设计,以确保程序逻辑正确并避免死循环。另外,由于 do-while 循环至少执行一次循环体,因此适用于需要先执行一次再检测条件的情况,例如用户输入验证等场景。由于 do-while 语句的运行逻辑与上文的两个循环语句相同,故在此只进行说明与讲解,不进行例题展示。

3.6 程序跳转语句

3.6.1 break 语句

在 C 语言中,break 语句用于从 switch 语句或循环结构(例如 for、while、do-while)中跳出,即使当前的循环条件没有满足。

当执行到 break 语句时,程序将立即跳出所在的循环或 switch 语句,并继续执行循环或 switch 语句后面的代码。使用 break 语句可以提高编程效率并加速程序的运行。

以下代码演示了如何在 for 循环中使用 break 语句:

```
for (int i = 0; i < 10; i++) {
    if (i == 5) {
        break;
    }
    printf("%d\n", i);
}
```

以上代码将循环输出数字 0～4，当 i＝5 时，遇到 break 语句，循环中断，程序将立即跳出循环体并继续执行循环后面的代码。

需要注意的是，break 语句只能用于跳出内部循环或 switch 语句，不能直接跳出外层循环或函数。如果需要在多重嵌套的循环中跳出多层循环，可以使用标签(label)配合 break 语句实现。

【例 3.7】　引导 Copilot 分析代码。

```
#include <stdio.h>

int main(){
    int i;
    for(i=1;i<=10;i++){
        printf("* ");
        if(i>=5)  break;
        printf("|");
    }
    printf("\ni=%d",i);
    return 0;
}
//请写出这段代码的运行结果
//
//* | * | * | * | *
//i=5
```

在这个例题中，首先给 Copilot 一段完整的可编译代码，要求 Copilot 在不编译运行的前提下对代码结果进行分析输出，所以引导的自然语言内容在最后放出。

3.6.2　continue 语句

在 C 语言中，continue 语句也用于控制循环结构，与 break 语句不同的是，它被用来跳过当前循环中剩余的语句，并进入下一次迭代，即当程序执行到 continue 语句时，会直接开始下一次循环迭代，而不再执行循环体中剩余的代码。

以下代码演示了如何在 for 循环中使用 continue 语句：

```
for (int i = 0; i < 10; i++) {
    if (i == 5) {
        continue;
    }
    printf("%d\n", i);
}
```

以上代码将循环输出数字 0～9，当 i＝5 时，遇到 continue 语句，程序将跳过当前迭代中的剩余代码，立即进入下一次迭代，继续执行后续的循环语句。

需要注意的是，在使用 continue 语句时，应确保它不会导致无限循环，否则程序可能会陷入死循环。因此，通常应将 continue 语句和某个 if 条件检查结合使用，以确保它只在特定条件下执行。

同时,在编写复杂的循环结构时,应注意 continue 语句对程序执行流程的影响,以保证程序逻辑正确并确保循环能够顺利结束。

【例 3.8】 分析以下代码并观察其与例 3.7 的运行结果的区别。

```
#include <stdio.h>

int main(){
    int i;
    for(i=1;i<=10;i++){
        printf("* ");
        if(i>=5)  continue;
        printf("|");
    }
    printf("\ni=%d",i);
    return 0;
}
//请写出这段代码的运行结果
//* | * | * | * |******
//i=11
```

可以看到,两个例题的运行结果因为 break 与 continue 语句的不同使用方式而产生了不同的结果,请读者自行对二者的区别进行分类分析,并加以练习。

3.6.3 goto 语句

在 C 语言中,goto 语句是用来实现跳转的一种语句。它可以使程序无条件地跳转到代码中的指定位置,而不考虑程序当前所处的执行状态。goto 语句容易使程序出现逻辑混乱和不可预测行为,因此应慎重使用。

以下代码演示了如何在 C 语言中使用带标签的 goto 语句:

```
#include <stdio.h>

int main() {
    int i = 0;
start:                                          //标签
    printf("%d\n", i);
    i++;
    if (i < 10) {
        goto start;                             //跳转到标签处
    }
    return 0;
}
```

以上代码使用了一个带标签的 goto 语句,在每次循环迭代结束时,通过 goto 语句无条件地跳转到标签处,重新开始下一轮的循环迭代,并输出变量 i 的值。需要注意的是,goto 语句可能导致程序的控制流难以理解和维护,因此应尽量避免使用,除非在特定情况下有明确的需求和足够的技术能力保证程序的正确性和可维护性。

【例 3.9】　利用 if-goto 语句求 1 到 100 的和。

引导 Copilot 生成代码的自然语言如下：

```
//利用 if-goto 语句求 1 到 100 的和
```

Copilot 生成的代码如下：

```
#include <stdio.h>

int main() {
    int i = 1, sum = 0;
    loop:
    sum += i;
    i++;
    if (i <= 100) goto loop;
    printf("%d\n", sum);
    return 0;
}
```

3.7　多重循环及其利用

多重循环是指在程序中嵌套多个循环结构的一种编程技巧。通过多重循环，可以遍历和处理多维数组、矩阵等数据结构，并实现复杂的算法逻辑。

在 C 语言中，多重循环通常使用 for 循环或 while 循环嵌套实现，其基本语法结构如下：

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
        //循环体
    }
}
```

以上代码演示了两层 for 循环嵌套的实现方式，其中，外层循环控制变量 i，内层循环控制变量 j，并在循环体中处理需要的逻辑。

多重循环的利用主要有以下几方面。

- **遍历和处理多维数组、矩阵等数据结构：**多重循环可以遍历并访问多维数组、矩阵等数据结构的各个元素，从而进行相应的计算、操作或转换。
- **实现复杂的算法逻辑：**多重循环可以嵌套实现复杂的算法逻辑，例如图像处理、模拟仿真、搜索算法、排序算法等，从而提高程序的效率和准确性。
- **嵌套实现条件判断和控制语句：**多重循环可以嵌套使用 if 语句、switch 语句、break 语句、continue 语句等条件判断和控制语句，从而对程序的控制流进行详细的处理和调整。

需要注意的是，多重循环可能导致程序的效率和可读性降低，因此在编写多重循环时应

尽量考虑程序的性能优化和代码的可读性,避免出现过度复杂或冗余的代码结构。

由于本节更多的是讲解循环的多重嵌套,其本身的逻辑与 for、while 循环语句相同,故不进行例题展示。

3.8 循环程序设计方法

循环程序设计方法是一种常用的编程思维方式,其通过使用循环结构实现逐次迭代和处理大量数据。循环程序设计方法可以帮助程序员提高程序的效率和可读性,同时减少代码的重复和冗余。

在循环程序设计中,需要注意以下几方面。

- **循环变量的初始化**:在循环开始前应对循环变量进行初始化操作,例如将计数器变量赋初始值为 0 或 1。
- **循环条件的设定**:循环条件是控制循环是否执行的关键因素。在循环开始时,需要对循环条件进行判断,并根据判断结果决定是否执行循环体及是否继续循环。
- **循环变量的更新**:循环变量的更新是循环程序设计中的一个重要步骤,当循环体执行完后,需要对循环变量进行自增、自减等操作,以便下一次循环的执行。
- **循环控制语句的使用**:在循环程序设计中,控制循环流程的语句主要包括 break 语句(跳出循环)、continue 语句(跳过当前循环)等,通过合理地使用这些语句,可以有效控制循环程序的执行流程,从而提高程序的效率和准确性。

通过合理运用循环程序设计方法,可以实现各种数据处理操作和算法逻辑,例如对数组、列表、字符串等数据结构的遍历、搜索、排序等。需要注意的是,在编写循环程序时应尽量保证代码的简洁性和可读性,并注意避免出现无限循环等编程错误。

3.8.1 迭代法

迭代法是一种基于逐步逼近的数值计算方法,它通过对函数或过程进行多次迭代计算逼近函数或过程的精确解。在循环程序设计中,迭代法通常实现为循环结构,这使得它在数值计算和复杂算法实现中具有广泛的应用。

迭代法的基本思想是从一个初始值开始计算出一个逼近解,并用该逼近解替换原方程中的未知量,进而获得下一个更为接近的逼近解。按照这一迭代方式不断重复计算和更新,最终可以得到目标函数的精确解或近似解。

在循环程序设计中,迭代法通常实现为 while 循环或 for 循环结构。例如,要实现一个计算平方根的迭代算法,可以采用如下代码实现:

```
double square_root(double n, double eps) {
    double x = n;                              //初始值
    while (fabs(x * x - n) > eps) {            //循环条件
        x = (x + n / x) / 2;                   //迭代公式
    }
    return x;                                  //返回结果
}
```

以上代码通过 while 循环结构实现了平方根计算的迭代算法。其中，初始值为 n，循环条件为|x^2－n| ＞ eps，每次迭代通过公式 x＝(x＋n/x)/2 计算得到更接近目标平方根的逼近解。

需要注意的是，在使用迭代法进行数值计算时，应确保迭代算法的收敛性和精度，并注意避免出现死循环等错误。同时，对于一些复杂的计算问题，还可以采用多重循环嵌套、递归等高级编程技巧实现各种数值计算和数据处理操作。

【例 3.10】　兔子的繁殖问题。

引导 Copilot 生成代码的自然语言如下：

```
//假设兔子在出生两个月后就有繁殖能力，一对兔子每个月能生出一对小兔子。如果所有兔子都
//不死，那么一年以后可以繁殖出多少对兔子？请编程计算
```

Copilot 生成的代码如下：

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int n;
    scanf("%d", &n);
    int a[1000] = {1, 1};
    for (int i = 2; i < n; i++) {
        a[i] = a[i-1] + a[i-2];
    }
    printf("%d\n", a[n-1]);
    return 0;
}
```

3.8.2　穷举法

穷举法(也称枚举法)是一种基于遍历和判断的计算方法，它通过枚举所有可能性寻找问题的解。在循环程序设计中，穷举法通常运用嵌套循环结构将问题的所有可能情况依次列举出来，然后逐个进行测试和比对，最终确定问题的解。

穷举法一般适用于求解规模较小的离散问题，如密码破解、全排列、组合等问题，其基本思想是将问题中所有可能的答案都枚举出来，再逐一检查答案是否符合条件，直到找到正确答案或者遍历所有可能性为止。

例如，要求解一个整数 n 的平方根，可以采用如下的穷举法实现：

```
int sqrt(int n) {
    for (int i = 0; i <= n; i++) {              //枚举变量 i
        if (i * i == n) {                       //判断是否为目标解
            return i;                           //返回结果
        }
    }
```

```
    return -1;                                    //未找到解
}
```

该算法通过循环枚举变量 i 对每组可能的解进行判断,直至找到目标解或枚举完所有情况且并未发现目标解。需要注意的是,在使用穷举法进行计算时,由于其需要枚举所有可能性,因此对于问题规模较大的情况,其运行时间和空间复杂度都会相应增加,因此在实际计算中还需要结合具体问题进行分析,采用其他更高效的算法技术提升程序性能。

【例 3.11】 水仙花数问题。

引导 Copilot 生成代码的自然语言如下:

```
//编写程序,利用穷举法输出 100~999 中的所有水仙花数
```

Copilot 给出的代码如下:

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int i, j, k;
    for (i = 1; i <= 9; i++) {
        for (j = 0; j <= 9; j++) {
            for (k = 0; k <= 9; k++) {
                if (i * 100 + j * 10 + k == i * i * i + j * j * j + k * k * k) {
                    printf("%d\n", i * 100 + j * 10 + k);
                }
            }
        }
    }
    return 0;
}
```

本章小结

本章内容涉及许多关于程序控制语句的知识,以下是要点知识小结。

(1) if 语句是一种基本的条件流程控制语句,用于根据指定条件的真假情况执行不同的代码块。语法格式:if (条件表达式){代码块},可以通过嵌套实现多层条件判断。

(2) if-else 语句在 if 语句的基础上扩展,增加了当条件为 false 时执行的代码块。语法格式:if (条件表达式){代码块 1} else {代码块 2}。

(3) switch 语句用于根据不同的条件执行不同的代码块。语法格式:switch (表达式){case 常量:代码块 break;default:代码块}。

(4) while 语句用于循环执行某些代码,直到指定条件不再满足为止。语法格式:while (条件表达式){代码块}。当循环次数不确定时使用。

(5) for 语句是一种常用的循环语句,可以方便地实现计数器循环和遍历数组等操作。

语法格式：for（初始表达式；条件表达式；增量表达式）{代码块}。

（6）do-while 语句与 while 语句类似，不同之处在于它至少会执行一次循环。语法格式：do {代码块} while（条件表达式）。

（7）多重循环结构是指在程序中嵌套使用多个循环进行复杂的流程控制。同时可以通过所述相关程序跳转的控制语句 break、continue 和 goto 等语句，在多重循环中实现特殊的程序控制。

（8）循环结构方法设计是指根据程序需要分析合适的循环结构形式，并合理地应用到程序中。需要根据具体问题进行分析和设计，选择合适的循环结构，并注意优化程序性能。

本章内容涉及众多知识点，需要读者积极动手练习，反复巩固。同时，在实际应用中，还需要根据实际情况灵活运用各种控制语句，以优化程序效率，提高开发效率。

课后习题

一、选择题

1. C 语言中，if 语句的格式为（　　）。

A. if condition statement　　B. if statement

C. condition if statement　　D. statement if condition

2. 下列中可以用于 switch 语句的表达式类型是（　　）。

A. char　　B. float　　C. double　　D. long double

3. 在 for 循环的圆括号中，通常包含（　　）语句。

A. 1 个　　B. 2 个

C. 3 个　　D. 取决于循环体内部需要

4. C 语言中，while 循环的语法格式为（　　）。

A. while expression {statement;}　　B. while statement (expression) {}

C. while (expression) {statement;}　　D. expression while {statement;}

5. 在下列情况下，最适宜使用 for 循环结构的是（　　）。

A. 需要反复执行某一段代码直到满足条件时才退出循环

B. 需要当一定条件满足时对一个变量进行计数并且在次数达到上限时退出循环

C. 仅需在保证条件满足的情况下不停地重复执行一段代码

D. 需要在某段时间内以固定的时间间隔重复执行一段代码

6. 下列中不是 C 语言中的逻辑运算符的是（　　）。

A. &&　　B. ||　　C. !　　D. ? :

7. 在以下循环结构中，可以强制退出循环并跳转到指定位置的是（　　）。

A. if　　B. while　　C. do-while

D. switch　　E. none of the above

8. 下列代码的输出结果是（　　）。

```
int i = 0, j;
for (j = 0; j < 5; j++) {
    if (j > 2) {
```

```
            break;
        }
        i++;
    }
    printf("%d\n", i+j);
```

A. 2　　B. 3　　C. 4　　D. 5

9. 下列关于 do-while 语句的说法中正确的是(　　)。

A. 先判断条件是否成立,再执行循环体

B. 循环体至少执行一次,无论条件是否成立

C. 用于执行固定次数的循环

D. 条件表达式只支持相等和不相等运算符

10. 在 goto 语句中,标号的作用是(　　)。

A. 标识可以跳转到该位置的语句

B. 告诉编译器将标号所在的代码段放在内存的哪个位置

C. 标识本循环中的 continue 或 break 要跳转到的位置

D. 以上都不是

二、编程题

1. 编写一个程序,让用户输入一个正整数 n,计算并输出 1～n 中的所有偶数之和。

2. 编写一个程序,使用 while 循环读取用户从键盘输入的整数,求出这些整数之和,直到用户输入 0 为止,并将结果输出。

3. 编写一个程序,要求输入任意一组年份和月份,输出该月份有多少天。要求考虑闰年情况。

4. 从键盘输入一个整数 n,计算并输出 1＋2＋…＋n 的值。

5. 输入一组正整数,求出其中的最大值和最小值。