选择结构

if 语句

在编程时，经常会碰到需要根据某个条件是否为真来决定执行哪个语句，这时候就需要用到 `if` 语句。`if` 语句的格式如下：

```
if (条件 A) {
    ......
}
```

也就是说，当条件 `A` 为真时，执行省略号的内容。实例如下：

```
#include <stdio.h>
int main() {
    int n = 5; if(n > 3) {
        n = 9;
        printf("%d\n", n);
    }
    return 0;
}
```

输出结果：

```
9
```

上面的实例判断 `n > 3` 是否为真，如果为真，就令 `n` 为 9，并输出 `n`。

`if` 语句当条件满足的时候会执行其中的内容，但如果当条件不满足的时候也有语句需要执行的话，则应当使用 `else`，即如下格式：

```
if (条件 A) {
    ......
} else {
    ......
}
```

这样，当条件 `A` 成立时就会执行第一个省略号中的内容，当条件 `A` 不成立时则执行第二个省略号中的内容。实例如下：

```c
#include <stdio.h>
int main() {
    int n = 2;
    if(n > 3) {
        n = 9;
        printf("%d\n", n);
    } else {
        printf("%d\n", n);
    }
    return 0;
}
```

输出结果：

```
2
```

如果省略号中的内容只有一个语句，那么我们可以去掉大括号，使得外观上简洁一些。不过这样做会使某些复杂情况的实际逻辑跟自己的想法出现偏差。所以，一般只有在明确不会出错的情况下才可以将大括号去掉。

另外，如果我们需要在 `else` 的分支下需要再根据某个条件来选择不同的语句，那么可以使用 `else if` 的写法，即：

```c
if (条件 A) {
    ......
} else if (条件 B) {
    ......
} else {
    ......
}
```

这样就会先判断条件 `A` 是否成立，如果不成立，则判断条件 `B` 是否成立，如果还不成立，才会执行最后一个省略号的内容。实例如下：

```c
#include <stdio.h>
int main() {
    int n = 2;
    if(n > 3) {
        n = 9;
        printf("%d\n", n);
    } else if(n > 2) {
        printf("%d\n", n + 1);
    } else {
        printf("%d\n",n);
    }
    return 0;
}
```

输出结果：

```
2
```

最后学习一个技巧。在 `if` 的条件中，如果表达式是 `!= 0` 或 `== 0`，那么可以采用比较简单的写法：

(1) 如果表达式是 `!= 0`，则可以省略 `!= 0`。实例如下：

```
#include <stdio.h>
int main() {
    int n = 0, m = 5;
    if(n) {
        printf("n is not zero!\n");
    } else {
        printf("n is zero!\n");
    }
    if(m) {
        printf("m is not zero!\n");
    } else {
        printf("m is zero!\n");
    }
    return 0;
}
```

输出结构：

```
n is zero!
m is not zero!
```

上面的代码中，`if(n)` 的写法其实就是 `if(n != 0)`，这里由于 `if` 条件语句接收的是括号中表达式的“真”或“假”，也即 1 或 0。而 `n` 本身作为一个整数，当 `n` 为 0 时，则相当于为“假”，当 `n` 不为 0 时，则相当于为“真”。因此直接在 `if` 中填写这种表达式就可以直接作为真假判断（例如我们填写 `n + m` 也是可以的，则是判断 `n + m` 是否为 0）。

(2) 如果表达式为 `== 0`，则可以省略 `== 0`，并在表达式前添加非运算符 `!`。实例如下：

```
#include <stdio.h>
int main() {
    int n = 0, m = 5;
    if(!n) {
        printf("n is zero!\n");
    } else {
        printf("n is not zero!\n");
    }
    if(!m) {
        printf("m is zero!\n");
    } else {
        printf("m is not zero!\n");
    }
    return 0;
}
```

输出结果：

```
n is zero!
m is not zero!
```

上面 `if(!n)` 的写法就等价与 `if(n == 0)`。前面介绍过，非运算符的作用是将后面的表达式值真假颠倒。因此由于 `if(n)` 表示 `if(n != 0)`，那么 `if(!n)` 就表示 `if(n == 0)`。

这两个小技巧可能初学者会不太适应，但是确实可以简化写法，学弟妹在读到相应的程序时应当能够看出这种写法的意思。

if 语句的嵌套

```
if(条件 A) {
    ......
    if(条件 B) {
        ......
    } else {
        ......
    }
    ......
} else {
    ......
}
```

按上面的代码，当条件 `A` 成立时，会进入执行其大括号内的语句，执行期间碰到另一个 `if` 语句，当条件 `B` 成立或非成立时执行不同的语句。实例如下：

```
#include <stdio.h>
int main() {
    int n = 3, m = 5;
    if(n < 5) {
        if(m < 5) {
            printf("%d\n", m + n);
        } else {
            printf("%d\n", m - n);
        }
    } else {
        printf("haha\n");
    }
    return 0;
}
```

输出结果：

```
2
```

switch 语句

`switch` 语句在分支条件比较多的时候会显得比较干练，但是在分支条件较少时用得并不多。`switch` 语句的格式如下：

```
switch(表达式) {
case 常量表达式 1:
    ......
    break;
case 常量表达式 2:
    ......
    break;
case 常量表达式 n:
    ......
    break;
default:
    ......
}
```

我们来看一个实例：

```
#include <stdio.h>
int main(){
    int a = 1, b = 2;
    switch(a + b) {
    case 2:
        printf("%d\n", a);
        break;
    case 3:
        printf("%d\n", b);
        break;
    case 4:
        printf("%d\n", a + b);
        break;
    default:
        printf("bad ending\n");
    }
    return 0;
}
```

输出结果：

```
2
```

上面这个实例中，我们以 `a + b` 作为需要判断的表达式：当 `a + b` 为 2、3、4 时各自有需要输出的东西，而其他情况则输出 `bad ending`。因为实际上 `a + b == 3`，因此选择 `case 3` 这条分支，输出了 `b`。另外，我们可以注意到，每个 `case` 下属的语句都没有使用大括号将它们括起来，这是由于 `case` 本身默认了把两个 `case` 之间的内容全部作为上一个 `case` 的内容，因此不用加大括号。

我们还应该注意到，每个 `case` 的最后一个语句都是 `break`。这个 `break` 有什么作用呢？我们不妨把所有的 `break` 都删掉，再输出结果看看：

```
```

```c
#include <stdio.h>
int main() {
    int a = 1, b = 2;
    switch(a + b) {
    case 2:
        printf("%d\n", a);
    case 3:
        printf("%d\n", b);
    case 4:
        printf("%d\n", a + b);
    default:
        printf("bad ending\n");
    }
    return 0;
}
```

输出结果：

```
2
3
bad ending
```

我们会发现，删去 `break` 后，程序把 `case 3` 以下的所有语句都输出了。是的，`break` 的作用在于可以结束当前 `switch` 语句，如果将它删去的话，程序将会从第一个匹配的 `case` 开始执行语句，直到其下面的所有语句都执行完毕才会退出 `switch`。