高级语言的“不确定性”格式

确定性的语句：

```
int x = 114514;
int y = 1919810;
```

不确定的语句

```
if(x >810){
    x++;
}
printf("%d",x);
```

~~黄老师觉得学到这里就把C语言学完了🤣，数据结构，指针。~~

称之为

选择性结构：

`if` :就是那样括号里面有布尔条件运算吗，若1则运行花括号内语句，若0不运行直接跳过，细节看后面杂项。

`else` 的作用，排除了多if条件重复符合，因高级语言的顺序执行而导致

> 例子：如果我考了90分以上称之为优秀，如果我考了80分以上我就是良好，如果我考了60分以上那就是及格，虽然口语上是这样讲但是实际上还是有语病，真正符合逻辑思维的还得需要 **否则** 来描述，这样也引出else if的作用

举上面的例子的原因是为了避免你去 **死记** 一些莫名其妙的规则，而是一种自然而然的方式。

实现上述例子

- 第一种方法： `if+else` 直接
- 第二种方法：限定 `if` 的更苛刻条件。
- 第三种方法（纯装逼）：用 `switch` 结构：下面写一下

- ```
  int grade = 98;
  switch (grade){
  case >= 90 : balabala;
  case >= 80 : balabala;
  case >= 70 : balabala;
  case >= 60 : balabala;
  default : balabala;
  }
  ```

这是你用if和else加上自然语言的思路运行的，但实际上这是完全不行的，语法上全是问题，下面一步一步来改。

首先先排除没问题的项，比如后面的打印语句：

```
int grade = 98;
switch (grade){
case >= 90 :
  printf("nb");
case >= 80 :
  printf("not bad");
case >= 70 :
  printf("haixing");
case >= 60 :
  printf("not good");
default :
printf("sb");
}
```

通过报错，就知道

a. `grade` 变量不应该出现这么多次（本身就是一个简洁的代码块，放那么多变属于违背初衷了）
b. `>=` 但又是一个 **二元运算符**，两边都要有值，属于矛盾中的矛盾了

所以先把 **二元运算符** 给直接删了，只放一个数字，那也不挺好的吗.jpg
然后通过这个改了，得到这个代码：
```

```
int grade = 98;
switch (grade){
case  90 :
  printf("nb");
case  80 :
  printf("not bad");
case  70 :
  printf("haixing");
case  60 :
  printf("not good");
default :
printf("sb");
}
```

虽然这段代码还是跑不对，但至少不会报错了。
不过由于 `<=` 没了，同理也没有了这个逻辑，实际上case后面是直接跟 `=` 的，要满足例子中的条件，那只能再 *另辟蹊径* 了

```
int grade = 98;
switch (grade / 10){
case  9 :
  printf("nb");
case  8 :
  printf("not bad");
case  7 :
  printf("haixing");
case  6 :
  printf("not good");
default :
printf("sb");
}
```

但是这段代码也陷入一个新的问题: **我擦怎么又出现输出一堆值的情况了**
所以又得和 `else` 之于 `if` 的情况，所以开发出了一个新的语句 `break` ,使得满足一个case后，输出完了不会再 **落** 下去

- 其实还有一个 `fall through` ,它允许多个 `case` 共用代码

```
int grade = 98;
switch (grade / 10){
case 10 :   //算是利用了这个“直落”的好处
case  9 :
  printf("nb");break;
case  8 :
  printf("not bad");break;
case  7 :
  printf("haixing");break;
case  6 :
  printf("not good");break;
default :
printf("sb");
}
```

但有一个抽象的，虽然这段代码中； `default` 和 `else` 看似是一个功能，都是类似“否则执行”，但实际上， `switch` 只是先遍历所有 `case` 当所有 `case` 都不匹配时，程序才会跳转到 `default` 。用 `default` 的英语本意（默认）会更好理解一些

因此， `default` 可以放在switch上的任何位置上，没有规定一定要放在哪里

~~估计C创作的时候，磁盘容量和屏幕限制才导致创造出switch case这么抽象的语句吧~~

`switch case` 再大量条件判断时，还是比较好用的，代码运行效率和可读性都比较好。

循环

> 想累加，但是不想写一堆代码，搞得自己心情很不爽，想县中了。

不用循环只能这样写了：

```
int i = 1
int result = i

i++;
result = result + i;
i++;
result = result + i;
i++;
result = result + i;
……
```

这样做有两个坏处

- 首先就是不知道加到多少项了，只能眼镜嗯数，太不计算机太不IT了。
- 第二个是打字打得手都痛了

解决第一个问题，加上 **条件判断语句** 给一个控制条件，免于自己的眼睛劳役之苦
可以这样，比如说我只想累加7次，那就

```
if(i < 7){
  i++;
  result = result + i;
}
if(i < 7){
  i++;
  result = result + i;
}
if(i < 7){
  i++;
  result = result + i;
}
if(i < 7){
  i++;
  result = result + i;
}
if(i < 7){
  i++;
  result = result + i;
}
if(i < 7){
  i++;
  result = result + i;
}
if(i < 7){
  i++;
  result = result + i;
}
if(i < 7){
  i++;
  result = result + i;
}
if(i < 7){
  i++;
  result = result + i;
}
if(i < 7){
  i++;
  result = result + i;
}
if(i < 7){
  i++;
  result = result + i;
}
if(i < 7){
  i++;
  result = result + i;
```

```
}
if(i < 7){
  i++;
  result = result + i;
}
if(i < 7){
  i++;
  result = result + i;
}
```

这样就不用数数直接copy了，但是，**手还是很痛啊！！！**,而且如果整多了，代码也不会很简洁啊！
所以， `while` 就被发明了，简化了 **相同的选择**，按黄老师的说法，本质上没有什么 **循环结构**，***而是选择结构的一种高效整合***。

比如说 `while`

```
while(条件语句){
    //若布尔返回值（**负数也行**） != 0 则执行下列语句如：
    printf("sb");
}       //否则跳过该花括号
```

现在引入一个用 `whilie` 实现累加的情景

```
int ans = 0;
int i = 1;
while(i <= 5){
  ans += i;
  i++;
}
printf("%d",ans);
```

可以发现core code是 `ans += i;` 来实现累加，而 int i = 1, i++全 是**打辅助的**，而且这些“辅助代码”，很重要但放在 `while` 里又显得太分散了，创造者想要创造一个新的函数来整合这个功能，于是就有了

---

for

```
//上面的while完全可以整合成如下for语句，完 全 等 价。
int ans = 0;
for(int i = 1;i <= 1; i++){
  ans += i;
}
```

- 由此便可知晓 `for(a;b;c)` 中，abc都可以在while完全替代，只要观察的好，完全可以整明白
- `b` 就是布尔条件的判断，和 `while` 里完全一致，可以完全用一些正常的值替代。 **如果啥都不写默认是1**
- `a` 就是先定义一个 **（也可以是若干个）**，其实写在这里主要就是为了代码的简洁和美观，你想写外面完全没问题的 ~~别写在后面就行子~~
- `c` 就是任意的语句，相当于 `while` 花括号里的**最后N步**，每一个语句 **用逗号** 隔开， `c` 不一定只会执行一条语句！不过可能C founder真的不是很喜欢 `;)` 这种形式，所以分号也必须得省

dowhile

先判断再做事改成了先做事再判断，至少执行一次，原理没什么好说的，但是格式还是要说一下的

```
do{
  ans += i;
  i++;
}while(i <= 5); 这个**分号**一定不要漏了啊啊啊啊啊啊啊
```

if 的杂项

- 省略大括号时，**只执行括号后下一条语句**，执行完后自动跳出，就只执行一条哈！。
- 如果 `if();` ————→直接忽略判断，执行后面的语句。
- 如果 `if(){};` ————→直接把条件判断和大括号里面全忽略了，执行后面的语句。(以上两条至少在GCC14.0的编译器里是这样的……)

- ```
  if(x >= 5){
    printf("%d",x)
  }else{
    printf("sb");
  }
  ```

  和下列代码一样吗?
  ```

```
if(x >= 5){
    printf("%d",x)
}
if(x < 5){
    printf("sb");
}
```

不一样！

你把大括号去了，语句再多加点else之类的就老实了

`else`：总之就是一定要跟着 `if` 的 **末尾**