切換導航 Farseerfc的小窩

- 繁體
- <u>简体</u>
- **English** 日本語

- **≜** About
- **≜** Links
- About
- <u>Links</u>
- \Box
- □ Import
- □ Life
- <u>□ Tech</u>

- <u>Import</u>
- <u>Life</u>
- <u>Tech</u>
- Q

搜索

- 搜索
- *y*

C++ Tricks 3.2 標號、goto,以及switch的 實現 <u></u>

3.2 標號、goto,以及switch的實現

goto語句及標號(label)是最古老的C語言特性,也是最早被人們拋棄的語言特性之一。像彙編語言中的jmp指令一樣,goto語句可以跳轉到同一函數體中任何標號位置:

```
void f()
{int i=0;
Loop: //A label
++i;
if(i<10)goto Loop; //Jump to the label
}</pre>
```

在原始而和諧的早期Fortran和Basic時代,我們沒

有if then else,沒有for和while,甚至沒有函數的概念,一切控制結構都靠goto(帶條件的或無條件的)構件。軟件工程師將這樣的代碼稱作"意大利麪條"代碼。實踐證明這樣的代碼極容易造成混亂。

自從證明了結構化的程序可以做意大利麪條做到的任何事情,人們就開始不遺餘力地推廣結構化設計思想,將goto像猛獸一般囚禁在牢籠,標號也因此消失。

標號唯一散發餘熱的地方,是在switch中控制分支 流程。

很多人不甚瞭解switch存在的意義,認為它只是大型嵌套if then else結構的縮略形式,並且比if語句多了很多"不合理"的限制。如果你瞭解到switch在編譯器內部的實現機制,就不難理解強加在switch之上的諸多限制,比如case後只能跟一個編譯期整型常量,比如用break結束每一個case。首先看一個switch實例:

switch (shape.getAngle())

```
case 3: cout<<"Triangle";break;
case 4: cout<<"Square";break;
case 0:case1: cout<<"Not a sharp!";break;</pre>
default: cout<<"Polygon";
}
仟何程序員都可以寫出與之對應的if結構:
int i= getAngle(shape);
if (i==3) cout<<"Triangle";
else if(i==4) cout<<"Square";
else if(i==0||i==1) cout<<"Not a sharp!";
else cout<<"Polygon";
看起來這兩段代碼在語義上是完全一樣的,不是
麽?
不!或許代碼的執行結果完全一樣,但是就執行效
率而言、switch版本的更快!
```

要了解爲什麽switch的更快,我們需要知道編譯器 是怎樣生成switch的實現代碼的: 首先,保留switch之後由{}括起來的語具體,僅將其 中case、default和break替換爲真正的標號: switch (getAngle(shape)) { _case_3: cout<<"Triangle";goto _break; _case_4: cout<<"Square"; goto _break;</pre> _case_0:_case_1: cout<<"Not a sharp!"; goto break; default: cout<<"Polygon";

隨後,對於所有出現在case之後的常量,列出一張 只有goto的跳轉表,其順序按case後的常量排列: goto case 0;

break:

}

```
goto _case_1;
goto case 3;
goto _case_4;
然後、計算case之後的常量與跳轉表地址之間的關
係、如有需要、在跳轉表中插入空缺的項目:
100105: goto _case_0;
100110: goto case 1;
100115: goto _default; //因爲沒有case 2,所以插
入此項以條轉到default
100120: goto _case_3;
100125: goto _case_4;
假設一個goto語句佔用5個字節,那麼在本例中,
goto的地址=case後的常量*5+100105
```

之後,生成跳轉代碼,在其餘條件下跳轉至 default,在已知範圍內按照公式跳轉,全部的實現 如下:

```
{
int i= getAngle(shape);
if (i<0||i>=5)goto _default;
i=i*5+100105; //按照得出的公式算出跳轉地址
goto i; //僞代碼,C中不允許跳轉到整數,但是彙編
允許
100105: goto case 0;
100110: goto case 1;
100115: goto default;
100120: goto case 3;
100125: goto case 4;
_case_3: cout<<"Triangle";goto _break;
_case_4: cout<<"Square"; goto _break;
_case_0:_case_1: cout<<"Not a sharp!"; goto
break;
```

```
_default: cout<<"Polygon";
break:
```

}

經過這樣處理整個switch結構,使得無論switch後的變量為何值,都可以通過最多兩次跳轉到達目標代碼。相比之下if版本的代碼則採用線性的比較和跳轉,在case語句很多的情況下效率極低。

由此,我們也可以知道,爲什麼case後跟的一定是編譯期整型常數,因爲編譯器需要根據這個值製作跳轉表。我們可以明白爲什麼case與case之間應該用break分隔,因爲編譯器不改變switch語句體的結構,case其本身只是一個具有語義的標號而已,要想跳出switch,就必須用break語句。