

# C到C++的升级

C++继承了所有的C特性

C++在C的基础上提供了更多的语法和特性

C++的设计目标是运行效率与开发效率的统一

## 区别

### 0x1 变量定义的位置

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5
6      int c = 0;
7
8      for (int i = 1; i <= 3; i++)
9      {
10         for (int j = 1; j <= 3; j++)
11         {
12             c += i * j;
13         }
14     }
15
16     return 0;
17 }
```

使用C89标准编译不通过提示如下:

```
1  root@ubuntu:~/exp/DT_CPP/part01# gcc 1.c --std=c89
2  1.c: In function 'main':
3  1.c:8:5: error: 'for' loop initial declarations are only allowed in C99 or C11
   mode
4      8 |     for (int i = 1; i <= 3; i++)
5         |         ^~~
6  1.c:8:5: note: use option '-std=c99', '-std=gnu99', '-std=c11' or '-std=gnu11' to
   compile your code
7  1.c:10:9: error: 'for' loop initial declarations are only allowed in C99 or C11
   mode
8     10 |         for (int j = 1; j <= 3; j++)
9         |             ^~~
```

说明只有在 C99 或者 C11 以及使用 g++ 编译器 均可以编译通过

```
root@ubuntu: ~/exp/DT_CPP/ × + v
root@ubuntu:~/exp/DT_CPP/part01# gcc 1.c --std=c89
1.c: In function 'main':
1.c:8:5: error: 'for' loop initial declarations are only allowed in C99 or C11 mode
   8 |     for (int i = 1; i <= 3; i++)
     |         ^~~
1.c:8:5: note: use option '-std=c99', '-std=gnu99', '-std=c11' or '-std=gnu11' to compile your code
1.c:10:9: error: 'for' loop initial declarations are only allowed in C99 or C11 mode
   10 |         for (int j = 1; j <= 3; j++)
     |             ^~~
root@ubuntu:~/exp/DT_CPP/part01# gcc 1.c --std=c99
root@ubuntu:~/exp/DT_CPP/part01# g++ 1.c
root@ubuntu:~/exp/DT_CPP/part01#
```

## 0x2 register 关键字

代码优化关键字，早期的C不会自动优化，则人工优化，才有了 `register` 关键字  
`register` 关键字 请求 编译器将 局部变量存储在寄存器中  
使用 `register` 关键字修饰的变量

1. C语言无法获取 `register` 变量的地址
2. 对于早期的C++编译器发现程序中需要取 `register` 变量的地址时，`register` 对变量的声明无效。  
如今的C++编译器会将 `register` 关键字忽略

## 0x3 全局变量的定义

在C语言中，定义多个同名的全局变量是合法的，多个同名的全局变量最终被链接到全局数据区的同一块地址空间上。  
在C++中，不允许定义多个同名的全局变量

```
1 #include <stdio.h>
2
3 int a;
4 int a;
5
6 int main(void)
7 {
8     return 0;
9 }
```

```
root@ubuntu: ~/exp/DT_CPP/ X + v
root@ubuntu:~/exp/DT_CPP/part01# cat 3.c
#include <stdio.h>

int a;
int a;

int main(void)
{

    return 0;
}
root@ubuntu:~/exp/DT_CPP/part01# gcc 3.c --std=c11
root@ubuntu:~/exp/DT_CPP/part01# g++ 3.c
3.c:4:5: error: redefinition of 'int a'
    4 | int a;
      |     ^
3.c:3:5: note: 'int a' previously declared here
    3 | int a;
      |     ^
root@ubuntu:~/exp/DT_CPP/part01#
```

## 0x4 struct 关键字

1. C语言中的 `struct` 定义了一组变量的集合
2. C语言中 `struct` 定义的标识符并不是一种新的类型
3. C++中的 `struct` 用于定义一个全新的类型

C

```
1 struct _tag_student
2 {
3     const char * name;
4     int age;
5 };
```

定义一组变量的集合，该集合名称为 `_tag_student`，它并不是一种新的类型，如果要将它当做类型来使用，必须要用 `typedef` 关键字来定义一个类型名。

C++

```
1 struct Student
2 {
3     const char * name;
4     int age;
5 }
```

在 C++ 中则是一个全新的类型 `Student`

```
typedef struct _tag_student Student;
struct _tag_student
{
    const char* name;
    int age;
};
```



```
struct Student
{
    const char* name;
    int age;
};
```

## C 和 C++ 中结构体的等价定义

以上C代码中，应使用关键字 `typedef` 定义一种新的类型 `typedef _tag_student Student`，否则还是一组变量的集合，如上图

```
root@ubuntu: ~/exp/DT_CPP/ x + v
root@ubuntu:~/exp/DT_CPP/part01# cat lesson_2-2.c
struct _tag_student
{
    const char * name;
    int age;
};

int main(void)
{
    _tag_student a = {"helo", 1};

    return 0;
}
root@ubuntu:~/exp/DT_CPP/part01# gcc lesson_2-2.c
lesson_2-2.c: In function 'main':
lesson_2-2.c:10:5: error: unknown type name '_tag_student'; use 'struct' keyword to refer to the type
 10 |     _tag_student a = {"helo", 1};
    |     ^
    |     struct
lesson_2-2.c:10:23: warning: initialization of 'int' from 'char *' makes integer from pointer without a cast [-Wint-conversion]
 10 |     _tag_student a = {"helo", 1};
    |
lesson_2-2.c:10:23: note: (near initialization for 'a')
lesson_2-2.c:10:31: warning: excess elements in scalar initializer
 10 |     _tag_student a = {"helo", 1};
    |
lesson_2-2.c:10:31: note: (near initialization for 'a')
root@ubuntu:~/exp/DT_CPP/part01# g++ lesson_2-2.c
root@ubuntu:~/exp/DT_CPP/part01#
```

把变量的集合当作类型来用，那么 `gcc` 编译不通过

## 0x5 函数定义

- 在 C 语言中:
  - `int f()` 表示 返回值为 `int`，可以接收任意个实参
  - `f(void)` 表示 返回值为 `int`，无参函数
- 在 C++ 中:
  - `int f()` 与 `int f(void)` 具有相同的意义
    - 表示返回值为 `int`，无参函数

```
1 #include <stdio.h>
2
3 f(i)
```

```
4 {
5     printf("__f__ i = %d\n", i);
6 }
7
8 g()
9 {
10    return 5;
11 }
12
13 int main(void)
14 {
15
16    f(2);
17    printf("g() = %d\n", g());
18
19
20    return 0;
21 }
```

```
root@ubuntu: ~/exp/DT_CPP/ x + v
root@ubuntu:~/exp/DT_CPP/part01# cat lesson_2-3.c
#include <stdio.h>

f(i)
{
    printf("__f__ i = %d\n", i);
}

g()
{
    return 5;
}

int main(void)
{
    f(2);
    printf("g() = %d\n", g());

    return 0;
}
root@ubuntu:~/exp/DT_CPP/part01# gcc lesson_2-3.c --std=c89
root@ubuntu:~/exp/DT_CPP/part01# ./a.out
__f__ i = 2
g() = 5
root@ubuntu:~/exp/DT_CPP/part01# |
```

以上代码在 `gcc` 正常编译且运行

## 思考

► `int f()` 与 `int f(void)` 有什么区别?

## 小结

1. C++ 更 **强调实用性**, 可以在任意地方声明变量
2. C++ 中的 `register` 只是一个兼容的作用

3. C++ 编译器能够更好的进行优化
4. C++ 中的任意标识符都必须显示的指明类型