C到C++的升级

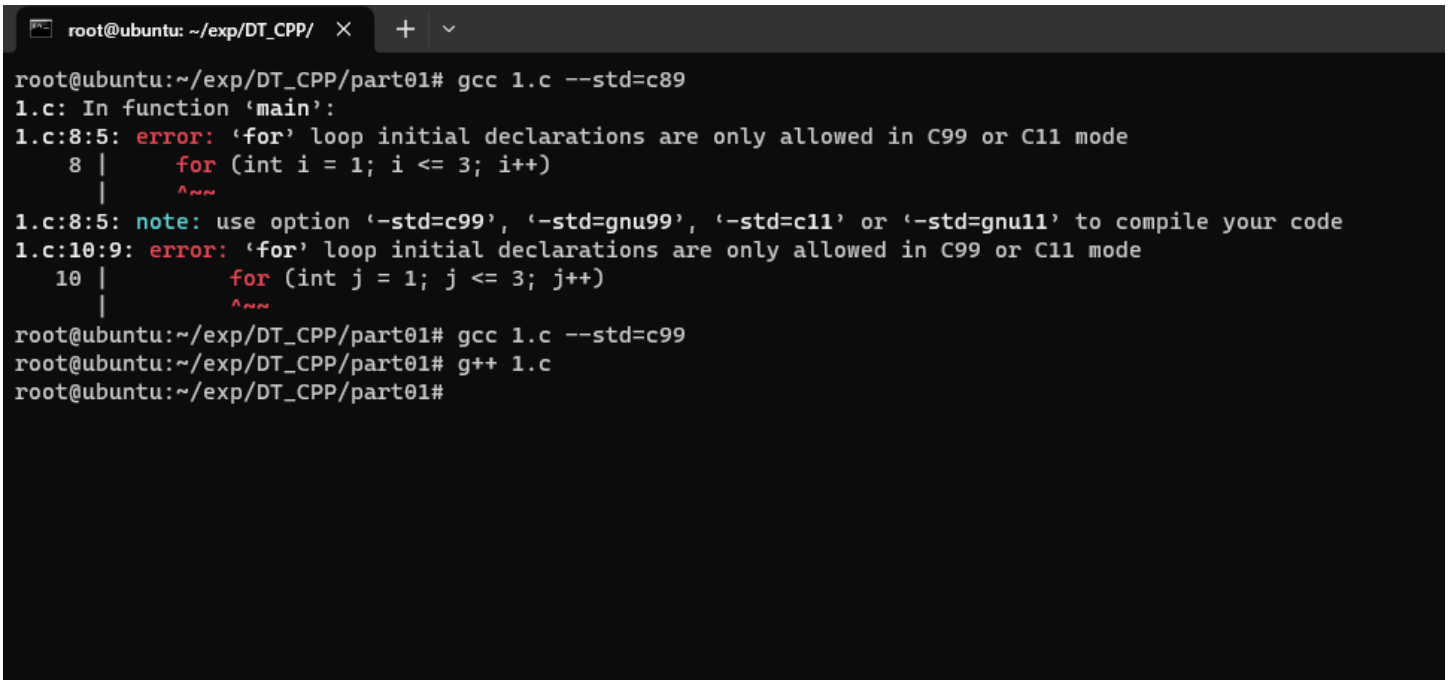
> C++继承了所有的C特性
> C++在C的基础上提供了更多的语法和特性
> C++的设计目标是运行效率与开发效率的统一

区别

0x1 变量定义的位置

```c
#include <stdio.h>

int main(void)
{

    int c = 0;

    for (int i = 1; i <= 3; i++)
    {
        for (int j = 1; j <= 3; j++)
        {
            c += i * j;
        }
    }

    return 0;
}
```

> 使用C89标准编译不通过提示如下：
>
> ```
> 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++)
>       |         ^~~
> ```
>
> 说明只有在 `C99` 或者 `C11` 以及使用 `g++` 编译器 均可以编译通过



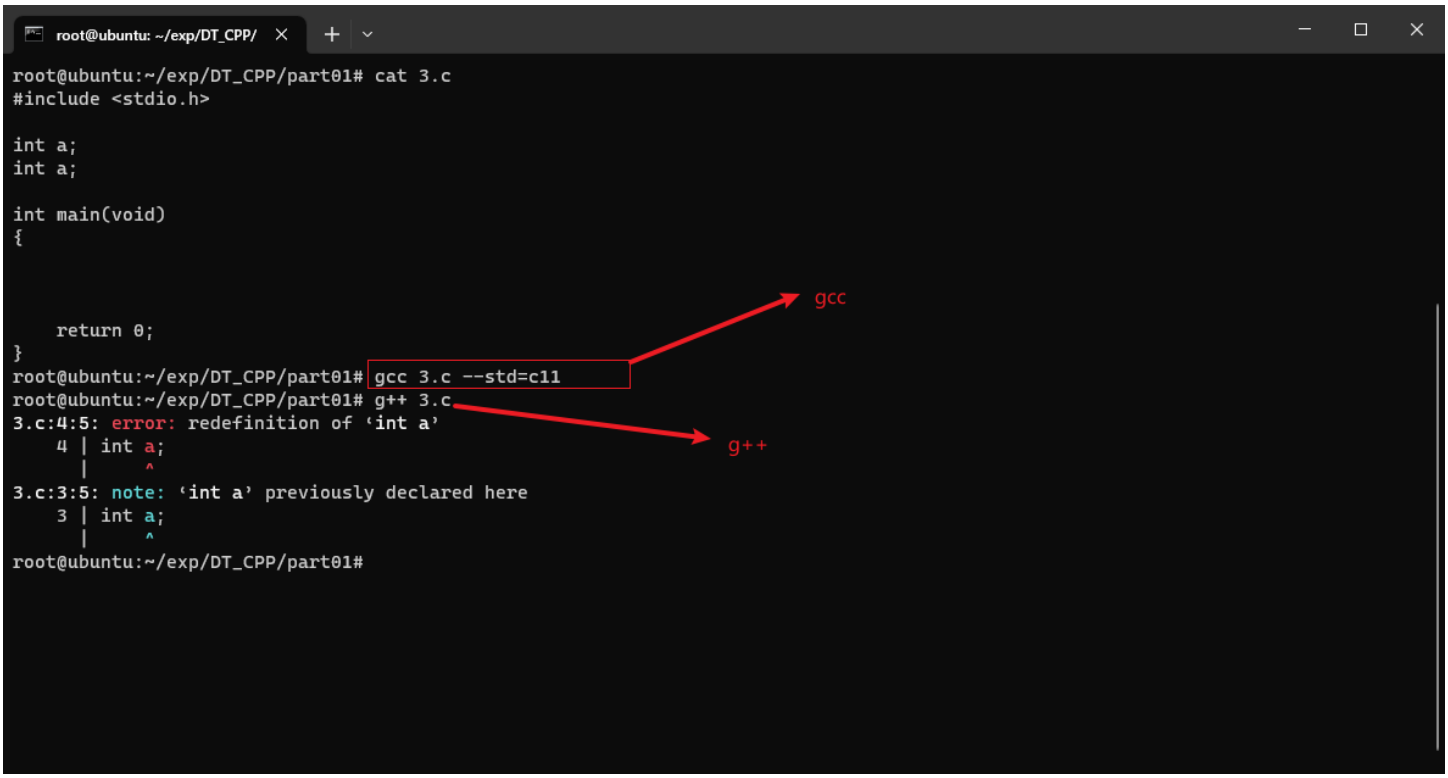
0x2 register 关键字

> 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

```c
#include <stdio.h>

int a;
int a;

int main(void)
{
    return 0;
}
```



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`

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

> 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++`

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

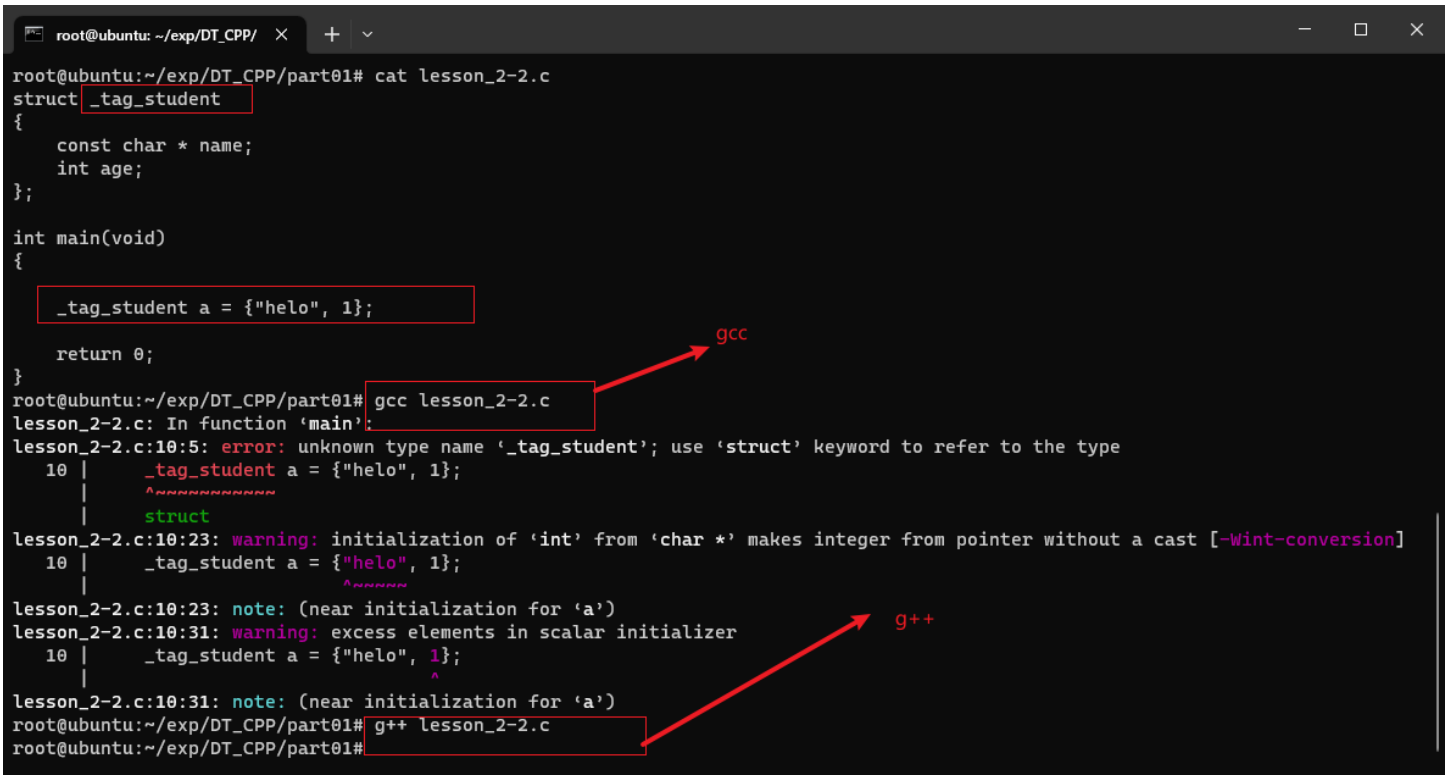
> 在 C++ 中则是一个全新的类型 `Student`

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

```c
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



> 把变量的集合当作类型来用，那么 `gcc` 编译不通过

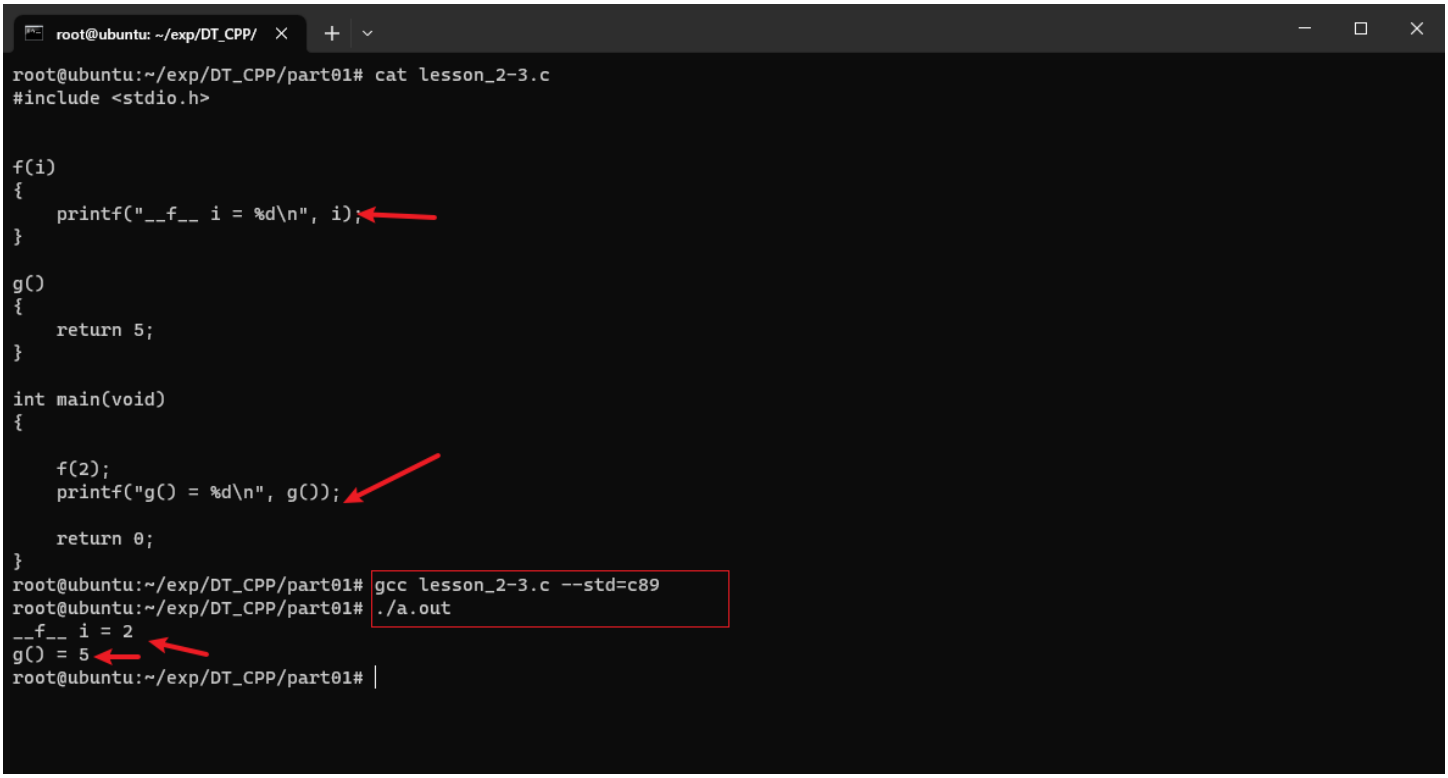
0x5 函数定义

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

    > 表示返回值为 `int`，无参函数

```c
#include <stdio.h>

f(i)
```

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

g()
{
    return 5;
}

int main(void)
{

    f(2);
    printf("g() = %d\n", g());


    return 0;
}
```



> 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