

# 课堂练习三

姓名	学号	日期
袁宇昊	201611130126	2018.11.20

## g++编译链接

以下测试程序源码含三个文件，编译链接中发现什么问题，把写出编译链接的结果信息。分析并说明问题原因并给出解决办法。

文件1: main.cpp

```
#include <iostream>
#include "add.cpp"
using namespace std;

int main()
{
    int a=1;
    int b=2;
    int c=0;

    c = add(a, b);

    cout << "c=" << c << endl;
    return 0;
}
```

文件2: add.hpp

```
int add(int a, int b);
```

文件3: add.cpp

```
int add(int a, int b)
{
    return a+b;
}
```

## 解答:

### 编译后的结果:

出现了main.o文件和add.o文件

(为什么不编译一下add.hpp文件?-->add.hpp文件是头文件, 不需要编译)

### 链接和的结果: 报错 多重定义了add函数

add.o:add.cpp:(.text+0x0): multiple definition of `add(int, int)' main.o:main.cpp:(.text+0x0): first defined here  
collect2.exe: error: ld returned 1 exit status

### 原因

在编译main.cpp函数的时候因为含有#include"add.cpp"命令行, 所以编译了一次add.cpp里的add函数, 当编译add.cpp时又编译了一次add函数, 所以当用链接命令g++.exe main.o add.o -o main.exe链接main和add时, 就相当于把add函数编译了两次, 程序里有两次add函数的定义, 所以报错。

### 修改后的正确编译链接得到可执行程序, 输出结果: ...

改法1: 把main.cpp函数里的#include "add.cpp"改为#include "add.hpp", 相当于在编译main.cpp时只用了add的声明, 没有实际定义, 所以没有实质编译add函数。

```
#include <iostream>
#include "add.hpp"//原来是"add.cpp"
using namespace std;

int main()
{
    int a=1;
    int b=2;
    int c=0;

    c = add(a, b);

    cout << "c=" << c << endl;
    return 0;
}
```

改法2: 把链接命令从g++.exe main.o add.o -o main.exe改为g++.exe main.o -o main.exe, 相当于直接把add.cpp函数里的东西写在main.cpp里。

```
C:\Users\HASEE\Desktop\C++\第八章多态\201611130126+袁宇昊+课堂练习03\test>"C:\Program Files
(x86)\CodeBlocks\MinGW\bin\g++.exe" main.o -o main.exe
```

## gdb调试过程

用 g++ 编译以下程序，并用 gdb 调试。

```
#include <iostream>
using namespace std;

int sub(int a, int b){
    return a-b;
}

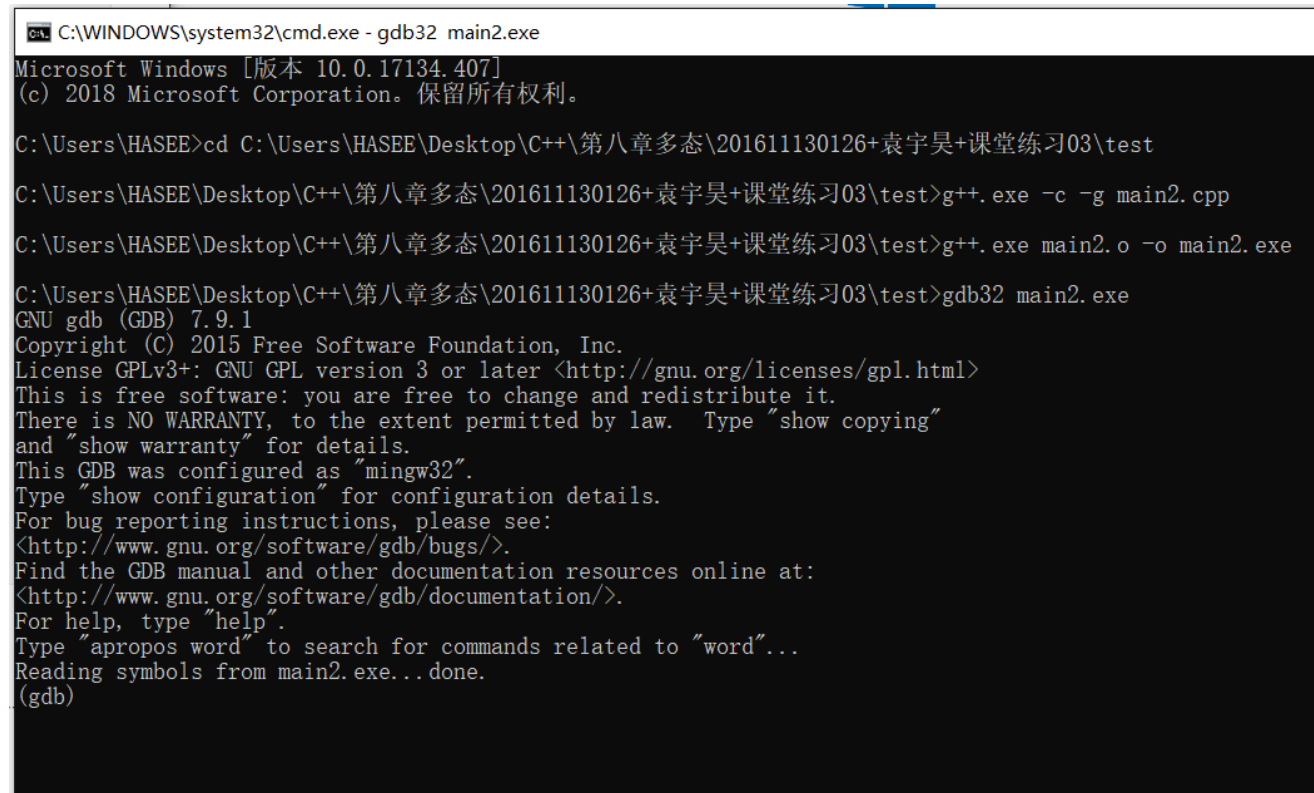
int main(void){
    int a = 10;
    int b = 8;
    int c = 0;

    c = sub(a, b);

    cout << c << endl;
    return 0;
}
```

## 解答：

先复制代码存在特定目录下的 main2.cpp 中，再用命令 `g++.exe -c -g main2.cpp` 去编译它，然后用命令 `g++.exe main2.o -o main2.exe` 生成 exe 文件，最后用命令 `gdb32 main2.exe` 去调试它。



```
C:\WINDOWS\system32\cmd.exe - gdb32 main2.exe
Microsoft Windows [版本 10.0.17134.407]
(c) 2018 Microsoft Corporation。保留所有权利。

C:\Users\HASEE>cd C:\Users\HASEE\Desktop\C++\第八章多态\201611130126+袁宇昊+课堂练习03\test
C:\Users\HASEE\Desktop\C++\第八章多态\201611130126+袁宇昊+课堂练习03\test>g++.exe -c -g main2.cpp
C:\Users\HASEE\Desktop\C++\第八章多态\201611130126+袁宇昊+课堂练习03\test>g++.exe main2.o -o main2.exe
C:\Users\HASEE\Desktop\C++\第八章多态\201611130126+袁宇昊+课堂练习03\test>gdb32 main2.exe
GNU gdb (GDB) 7.9.1
Copyright (C) 2015 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "mingw32".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from main2.exe...done.
(gdb)
```

然后就进入了 gdb 的调试界面：

如果直接输入 `r` 命令的话，会直接运行到结束。所以在第九行 `main()` 函数行 设置一个断点 `break 9`。

然后输入 `r` 命令开始运行程序。

多次使用n命令进行下一行，注意到n命令完成后，指向的是新的未完成的一行。

然后也使用了p x命令来查看x的值。注意p x命令的输出形式是\$s = x，s是一个数字，表示第几次调用p命令，x是查看到的值。我想这样写可以方便ide接受信息。

```
Breakpoint 1, main () at main2.cpp:9
9          int a = 10;
(gdb) n
10         int b = 8;
(gdb) p b
$1 = 7208852
(gdb) n
11         int c = 0;
(gdb)
```

最后使用了bt命令，查看函数堆，输出如下图：

```
(gdb) bt
#0  main () at main2.cpp:11
```

只有一个函数被调用，放在最下层，#0位置，并且记录了当前行数11。就好比栈的最底层。可以想见新调用的函数直接放在这个栈顶并记录行数。每次执行栈顶函数的下一行命令，知道return，递归。这样就完成了函数的调用。