

# Project3 总结

## ● 知识点

- 在 C 语言和 CPP 的混合编译时，经常会用到 `extern "C"` 这种语法，这是因为 CPP 具有多态性，支持函数的重载，CPP 对全局函数的处理方式与 C 有明显的不同。`extern "C"` 的主要作用是为了能够正确实现 CPP 代码调用 C 代码。加上 `extern "C"` 后，会指示编译器这部分代码按 C 语言进行编译，而不是 CPP。因为 CPP 支持函数重载，因此 CPP 的编译模式是编译函数的过程中会将函数的参数类型加到编译后的代码中，而不仅仅是函数名；而 C 不支持函数重载，因此不会将参数类型加到编译后的代码后面，一般只包括函数名。

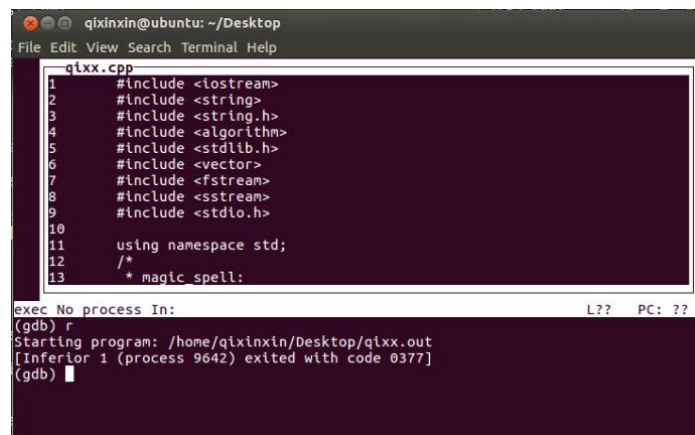
```
1  #ifndef __INCvxWorksh /*防止该头文件被重复引用*/
2  #define __INCvxWorksh
3
4  #ifdef __cplusplus    //_cplusplus是cpp中自定义的一个宏
5  extern "c" {          //告诉编译器，这部分代码按C语言的格式进行编译，而不是C++的
6  #endif
7
8      /**** some declaration or so ****/
9
10 #ifdef __cplusplus
11 }
12 #endif
13
14 #endif /* __INCvxWorksh */
```

- 一些代码优化的方法：方法之一是使用 `inline` 函数，它会大大提高运行速度，但会使得代码存储空间大大增加；方法之二是在使用 `g++` 的时候使用 `-O0`、`-O1`、`-O2`、`-O3` 编译优化选项，其中 `-O0` 能够将变量在寄存器中分配，实现循环的轮转，消除未使用的代码，简化表达式和声明并且调用声明为 `inline` 的函数；`-O1` 除了包含 `O0` 的所有优化选项之外，还能去除局部未使用的声明和局部公共表达式；`-O2` 除了包含 `O1` 的所有优化选项之外，还包含循环的优化，消除全局公共表达式和全局未使用的声明；`-O3` 除了包含 `O2` 的所有优化选项，还包含去除所有没有调用的函数，简化从未使用函数的返回值，使用 `inline` 方法调用小规格函数，重新排列函数的声明；

- GDB 调试：为了对代码使用 GDB 调试，需要在 `g++` 编译链接的时候添加 `-g`，打开方法为：`g++ -g qixx.cpp -o qixx`      `gdb qixx`

常用命令是：

- ◆ 使用 `Ctrl+X+A` 显示图形化界面，即能够显示代码



```
qixinxin@ubuntu: ~/Desktop
File Edit View Search Terminal Help
qixx.cpp
1  #include <iostream>
2  #include <string>
3  #include <string.h>
4  #include <algorithm>
5  #include <stdlib.h>
6  #include <vector>
7  #include <fstream>
8  #include <sstream>
9  #include <stdio.h>
10
11  using namespace std;
12  /*
13  * magic spell:
14
15  exec No process in: L?? PC: ??
(gdb) r
Starting program: /home/qixinxin/Desktop/qixx.out
[Inferior 1 (process 9642) exited with code 0377]
(gdb)
```

- ◆ r 命令启动程序 (run)
  - ◆ b 添加断点, 清除、禁用、启用断点分别为 delete、disable、enable;
  - ◆ 单步执行 s (step): 遇到函数会进入
  - ◆ 单步执行 n (next): 遇到函数不会进入
  - ◆ 执行下一断点 c (continue)
  - ◆ 查看变量 p (print)
  - ◆ 显示变量 display
  - ◆ 查看当前调用堆栈 bt (backtrace)
  - ◆ 查看某一层调用代码 f (frame)
  - 时间度量: time 指令
  - 性能损耗: perf record 指令 perf report
- 还有一些重点: string 用 char\* 表示、容器尽量少用, 因为底层涉及很多函数调用, 因此耗时很多。

## ● 个人总结

这次的 project 做的不好, 出现的问题主要有两个, 一个是内存的使用: 之前这种深搜、建树的题目每一次都是用了指针之后不再管它, 但可能是之前的数据比较少, 所以不会导致内存溢出。这次的时候随便跑一个比较长的数据集的时候, 电脑总是死机 (windows 下), 每次跑代码的时候打开任务管理器就会看到内存直线上升, 我甚至不太敢放到服务器上去跑, 后来查了一下并且验证之后发现确实是我指针使用的问题, 用了之后没有 delete 掉, 而且 delete 之后要将指针置为 NULL; 另一个问题是代码本身的问题: 开始的时候希望匹配字符串的算法是最有效率的 (因为大家普遍的字符串匹配算法是 KMP), 所以使用了一种算法——sunday 算法。这种算法比较好理解, 并且效率比 KMP 更高, sunday 算法的基本思路是: 假设子串为 s, 长度为 lens, 母串为 m, 首先从头开始逐个字符匹配, 当出现失配字符的时候, 就判断母串的 lens+1 处的字符是否在母串中存在, 如果存在的后子串后移一位, 否则, 重新从 lens+2 处开始匹配整个子串。我觉得这个算法本身并没有问题, 只是在实际运行的时候速度太慢 (没有剪枝的原因??) 不过用了编译优化的 O3 之后时间明显减少为原来的三分之一。