**汇编语言程序设计**

**大作业**

C/C++语言的秘密

学院：计算机科学与技术学院

专业：信息安全

班级：1503202班

学号：1150320202

姓名：刘佳斌

2016年11月20日

1. 概述

汇编语言是程序设计语言中，最接近机器语言，可以直接硬件的编程语言。我们知道，C语言或者C++语言编写的程序，要被翻译成一条条汇编指令，最后才能转换为可执行文件，可见汇编语言的重要性。

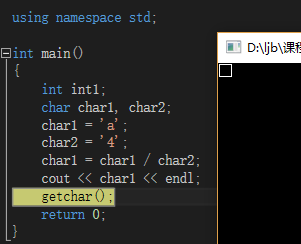
通过对汇编语言的学习，我们不仅学习到了使用汇编语言编写程序，完成某一功能的方法，更重要的是，我们学到了很多和硬件相关以及和软件相关的知识。在这门课程的学习过程中，我们对计算机处理器的发展有了大致的了解，而且了解了不同时代的处理器是如何管理内存，如何组织和存储数据等等；同时，我们也理解了高级语言的很多语法，数据等在汇编层面是如何实现的，学习到了在机器语言层面调试程序的方法，加深了对高级语言的理解。另外，我们还学到了一些计算机的基础知识，比如一些布尔概念，数据的表示方法等。

我将在这篇文章中对汇编语言和C++语言做一个总结和对比，以此加深对本课程的理解。

1. C++语言和汇编语言的语言元素
2. C++的语言元素

（1）数据类型

|  |  |  |  |
| --- | --- | --- | --- |
|  | 数据类型 | 标识符 | 所占字节数（位数） |
| 整型类 | 短整型 | short[int] | 2(16) |
| 整型 | int | 4(32) |
| 长整型 | long[int] | 4(32) |
| 超长整型 | long long[int] | 8(64) |
| 无符号短整型 | unsigned short[int] | 2(16) |
| 无符号整型 | unsigned int | 4(32) |
| 无符号长整型 | unsigned long[int] | 4(32) |
| 无符号超长整型 | unsigned long long [int] | 8(64) |
| 实型 | 单精度实型 | float | 4(32) |
| 双精度实型 | double | 8(64) |
| 长双精度实型 | long double | 16(128) |
| 字符型 | 字符型 | char | 1(8) |
| 布尔型 | 布尔变量 | bool | 1(8) |
| 无值型 | 无值型 | void | 0 |
| 字符串型 | 字符串型 | string | —— |
| 指针型 | 指针型 | (TYPE)\* | 4(32) |
| 构造类型 | 数组 | —— | —— |
| 结构体 | struct | —— |
| 共用体 | union | —— |
| 枚举类型 | enum | —— |

C++的数据类型可以用来定义变量，常量，也可以利用基本数据类型生成构造数据类型。每一种数据类型在内存中所占的字节数是不同的，而且在不同的操作系统，不同的计算机上，同一种数据类型所占的字节数也不一定相同。

对不同的数据类型，在这种数据集上定义的操作就不相同，所以在使用C++编程时，程序员需要时刻注意自己所使用的变量的数据类型，比如像右图所示，虽然程序可以执行，但是我们得到了一个奇怪的结果。所以编译器在编译程序的过程中，会对变量的数据类型进行检查。

C++中的变量名，是对内存中某一内存单元的标识，使程序员不必记忆物理地址。

C++的字符串类型实质上是一个类类型，bool型用true表示“真”（1），用false表示“假”（0）。这两个数据类型是C++中特有的，C语言中没有。

下面着重对四种构造数据类型进行讨论。

数组，就是相同数据类型的元素按一定顺序排列的集合，就是把有限个类型相同的变量用一个名字命名，然后用编号区分他们的变量的集合，这个名字称为数组名，编号称为下标。组成数组的各个变量称为数组的分量，也称为数组的元素，有时也称为[下标变量](http://baike.baidu.com/subview/11753436/12121379.htm)。数组是在程序设计中，为了处理方便， 把具有相同类型的若干变量按有序的形式组织起来的一种形式。这些按序排列的同类[数据元素](http://baike.baidu.com/subview/38785/38785.htm)的集合称为数组。

结构体，是一个可以包含不同数据类型的一个结构，它是一种用户可以自定义的数据类型。它的特点和数组主要有两点不同，一是结构体可以在一个结构中声明不同的数据类型，二是相同结构的结构体变量是可以互相赋值的。结构体一个最终要的功能就是将逻辑上有联系的不同数据类型组织在了一起，这在编程中是十分有用的。C++在结构体的基础上出现了类，将数据和结构进行了封装，使用户可定义自己的数据类型。

共用体类型也是用来描述类型不相同的数据，但与[结构体类型](http://baike.baidu.com/view/4291813.htm)不同，共用体[数据成员](http://baike.baidu.com/view/709620.htm)存储时采用覆盖技术，共享(部分)[存储空间](http://baike.baidu.com/view/1375172.htm)。在[结构体](http://baike.baidu.com/view/204974.htm)中增加共用体类型成员，可使结构体中产生动态成员，语言的变体记录。共用体类型在有的书中亦译为[联合体](http://baike.baidu.com/view/39140.htm)类型。

枚举类型C++中的一种派生数据类型，它是由用户定义的若干枚举常量的集合。如果一个变量只有几种可能的值，可以定义为枚举类型。所谓“枚举”是指将变量的值一一列举出来，变量的值只能在列举出来的值的范围内。

（2）宏

C++[宏定义](http://baike.baidu.com/view/2076445.htm)将一个[标识符](http://baike.baidu.com/view/390932.htm)定义为一个字符串，[源程序](http://baike.baidu.com/subview/546605/546605.htm)中的该标识符均以指定的字符串来代替。C++使用宏可以定义宏常量，将某些经常使用的值定义为一个宏，方便调试。宏也可以用来作为条件编译的标识，例如#define和#ifdef配合使用，在头文件中防止文件被反复包含。使用宏还可以用来定义宏函数，宏函数可以用来避免函数调用，提高程序效率。

C++中的宏功能强大，有很多用处。

（3）运算

C++的运算大致可以分为赋值运算，算术运算，逻辑运算，关系运算等。

赋值运算就是使用算符运算对变量进行赋值，如果赋值运算符两侧的类型不一致，但都是数值型或字符型时，在赋值时会自动进行类型转换。

算数运算就是使用算数运算符+、-、\*、/、%，自增自减运算符++、--，复合运算符+=、-=、\*=、/=等，对整型，实型等进行运算。用算术运算符和括号将运算对象（也称操作数）连接起来的、符合C++语法规则的式子，称为算术表达式，运算对象包括常量、变量、函数等C++语言规定了运算符的优先级和结合性。在求解表达式时，先按运算符的优先级别高低次序执行，例如先乘除后加减。C++规定了各种运算符的结合方向（结合性），算术运算符的结合方向为“自左至右”，即先左后右

逻辑运算就是使用&&、||、！等进行的运算，往往要求根据某个指定的条件是否满足来决定执行的内容。

关系运算，实际上是比较运算，使用关系运算符<、>、==、!=、<=、>=等将两个数据进行比较，判断比较的结果。用关系运算符将两个表达式连接起来的式子，称为关系表达式。

（4）子程序

对C++而言，其子程序主要是指函数。

函数就是一段实现某个功能的代码。一个程序包含若干个函数，但只有一个是主函数，程序从主函数开始执行，执行过程中，主函数调用其他函数，其他函数互相调用。使用函数，可以减少代码的重复编写，提高程序的重用性，可维护性等。函数如果在调用之前定义，则可以省略函数的声明，否则要先声明后定义。我们可以调用标准库函数，也可以自定义函数，然后调用。

1. 汇编语言的语言元素
2. 数据类型

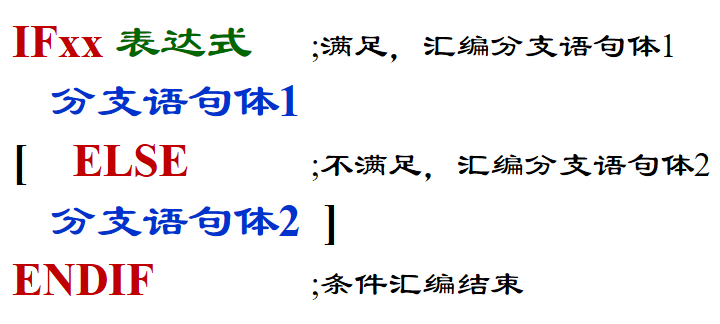
|  |  |  |
| --- | --- | --- |
| 数据类型 | 标识符 | 所占字节数（位数） |
| 8位无符号整数 | BYTE | 1(8) |
| 8位有符号整数 | SBYTE | 1(8) |
| 16位无符号整数 | WORD | 2(16) |
| 16位有符号整数 | SWORD | 2(16) |
| 32位无符号整数 | DWORD | 4(32) |
| 32位有符号整数 | SDWORD | 4(32) |
| 48位整数 | FWORD | 6(48) |
| 64位整数 | QWORD | 8(48) |
| 80位整数 | TBYTE | 10(8) |

在汇编语言中，(S)BYTE被称为字节，(S)WORD被称为字，(S)DWORD被称为双字。这三种数据类型在汇编语言中最常用，可以用这些数据类型来定义变量。也可以用来定义数组，有趣的是，汇编语言中定义的数组每个元素只需要占用的字节数相同即可，例如语句“array BYTE 1,2,’A’,-1,’<’，0”，可以是字符、数字的组合。汇编语言中的变量名实质上就是一个标号，标识着这个变量在数据段中的偏移量。每一种数据类型既可以存储整数，也可以存储字符等，还可以对变量定义别名，使用非常灵活。

1. 宏

在汇编语言中，宏汇编，重复汇编和条件汇编统称为宏结构。宏是汇编语言的一个特点，它是与子程序类似又独具特色的另一种简化源程序的方法。

宏是具有名字的一段汇编语句序列；宏指令是这段汇编语句序列的缩写；宏展开是指在宏指令处插入宏的代码副本。宏定义一定要写在程序的开始处，即数据段和代码段前，或INCLUDE指令的文件中。在程序中，宏调用的实质就是在在汇编过程中进行宏展开。

汇编中的宏定义，是以有图所示的方式定义的，宏定义中可以有任意多的参数，语句列表中的语句直到宏被调用的时候才会被编译。还有许多与宏有关的伪指令和宏操作符在宏定义和调用的过程中可以使用。宏嵌套，一个宏的定义中调用了另一个宏。调用嵌套宏时，传递给外层的参数可直接传递给内层的宏。宏函数也是一个命名的汇编语句块，唯一的不同在于，宏函数总是用EXITM伪指令返回一个常量值(整数或字符串)，宏函数在调用时参数列表必须用圆括号括起来。条件汇编伪指令在汇编过程中，根据条件决定汇编的语句，如右图所示。重复汇编指在汇编过程中，重复展开一段（基本）相同的语句重复汇编没有名字，不能被调用重复汇编常用在宏定义体中，也可以在一般汇编语句中使用。重复汇编的伪指令有三个，REPEAT，FOR，FORC，最后用ENDM结束。

1. 运算

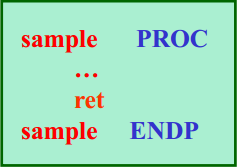
在汇编语言中，算术运算不是使用运算符来完成的，而是使用操作数。

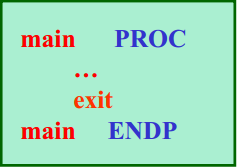
加法指令ADD，将同尺寸的目的操作数和源操作数相加，将结果保存在目的操作数中。减法指令SUB，是将源操作数从目的操作数中减掉。此外，还有NEG指令，将数字转换为对应的二进制补码,从而求得其相反数。INC（DEC）指令从操作数中加1（减1）。汇编语言中的乘法和除法指令比较复杂。MUL是无符号乘法指令，它是一条单操作数指令，被乘数、积由乘数隐含制定。IMUL是有符号数的乘法指令，有单操作数和多操作数两种格式。无符号数的除法指令DIV，是一条单操作数指令，被除数，商及余数都由除数的大小决定。IDIV是有符号数除法指令，指令格式与DIV相同。

汇编语言中的逻辑运算，是由逻辑运算指令AND，OR，XOR，NOT来完成的。其中，AND，OR，XOR在每对操作数的对应数据位之间执行布尔位“与”、“或”、“异或”操作，并将结果保存在目的操作数中。NOT指令则是将一个操作数的所有数据位取反。

关系运算由CMP指令来完成。CMP指令执行隐含的减法操作，根据减法的结果设置标志位，但是不保存减法的结果，利用标志位，就可以判断两个操作数的大小。

1. 子程序

汇编语言中的函数，使用PROC和ENDP伪指令来定义，除了启动过程之外的其他过程都应以RET指令结束，启动过程以exit语句结束，如下图所示。



函数调用使用CALL指令，CALL指令将返回地址压入堆栈并将被调用过程的  
地址拷贝到指令指针寄存器中；设置处理器在新的内存地址执行指令，实现对过程的调用。当程序返回时，RET指令从堆栈弹出地址并送到指令指针寄存器中；使处理器返回到程序中调用过程的地方继续执行。形参和返回值可以通过寄存器传递，形参还可以使用堆栈来传递。函数的调用还可以使用INVOKE伪指令，INVOKE伪指令可以传递多个参数。

3.各元素的对应关系

（1）数据类型

C++中的基本数据类型int，char，short，unsigned int等，都是对应汇编中的byte，word，dword这些基本类型。比如int就对应sdword，char对应byte。C++中占相同字节数的元素，实质上与汇编对应的数据类型都是相同，只不过C++的编译器会对数据类型进行严格的检查。这在汇编层面看，是相同的。

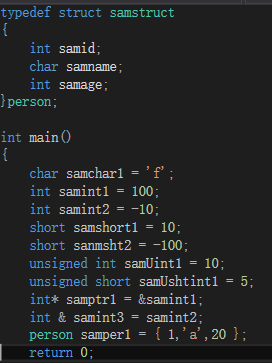
C++和汇编中都有宏的概念，二者很类似，比如都可以定义一些宏函数，进行条件编译。宏调用的时候，都是替换。

C++中的算术运算是通过+、-、\*、/、%等操作符来完成的，这些都对应的汇编中的算术运算指令，add，sub，mul/imul，div/idiv。不过，在汇编中，乘法和除法要比C++中复杂的多，更有意思的是，C++中的除法和求余操作，对应的都是汇编语言的div/idiv这一条指令。C++中的逻辑运算是使用&&、||、！等逻辑运算符实现的，对应语汇编中是使用逻辑运算指令and，or，not，xor来完成的，这些汇编指令都是按位来操作的，当然，在C++中也对应的位运算符。C++的关系运算是使用关系运算符>、<、==、!=、<=、>=等来完成的，这些对应汇编中，都是cmp指令，通过cmp指令设置的标志位来判断大小。

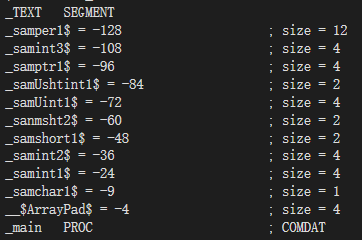
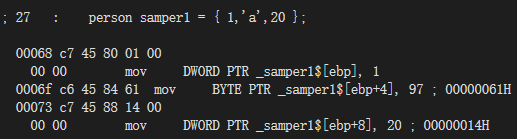
C++的函数和汇编的函数也是非常类似的，这里不做详细讨论，下文中会结合C++程序和其反汇编代码具体分析。

1. C++语言的反汇编分析

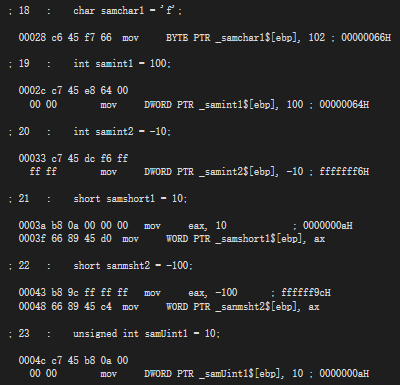
接下来，我将使用C++编写一些程序，结合其反汇编代码，进行深入分析。

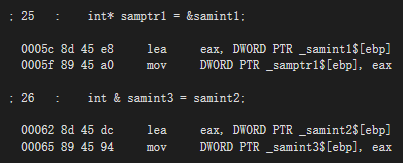
1. 常用数据类型的分析

如右图所示，我使用C++定义了一个结构体person，在main函数中定义了一些变量，还有一个使用person这个结构体定义的变量。x下面是这段代码反汇编中的一部分，展示了每一个变量在内存中占用的字节数。

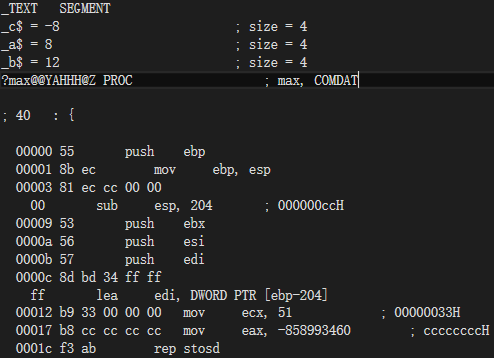
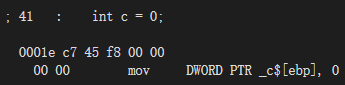
基本数据类型所占的字节数我们在上文中已经做过讨论，我们看一下结构体变量samper1，占用了12个字节，但是我们看结构体的声明，两个int成员和一个char成员，按常理，占用字节数4+4+1=9，和实际12个字节不相同。这是为什么？实际上，为了达到最佳的IO效果，将每一个数据成员都对齐到了4字节的整数倍的地址上，也就是说，samname的后面三个字节没有被使用，跳过这三个字节，然后在存储samage，就像下面这样。数据成员在内存中的先后顺序，和我们的定义是对应的。

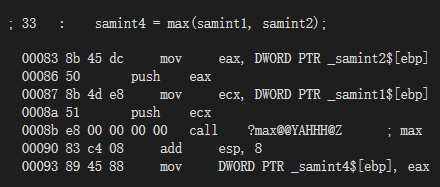
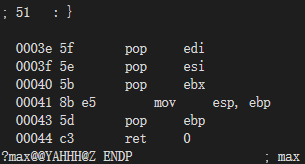
|  |  |  |  |
| --- | --- | --- | --- |
| samid | | | |
| samname |  |  |  |
| samage | | | |

对语普通的变量，我们可以看到它们都是根据所占内存的字节数，和汇编中的byte，word，doword相对应。普通变量的赋值，都是使用寄存器ebp和一个偏移值来寻址相应的内存单元，然后将值写入。在这里我们还可以看到内存中的是都是使用补码来存储的。

对于指针变量samptr1而言，它的实现是先使用lea指令，将其所指向的内存单元的地址值存储在寄存器eax中，然后将eax的值送入samptr1所指向的内存单元。对于引用类型samint3而且，在汇编层面，其实现过程语指针变量完全相。可见，这二者的区别并不是汇编层面上的，而是在C++的编译器中加以区别的。

1. 函数、参数、局部变量的分析

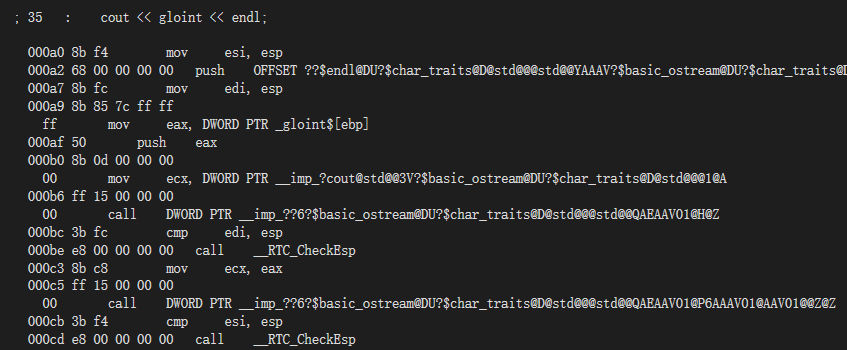
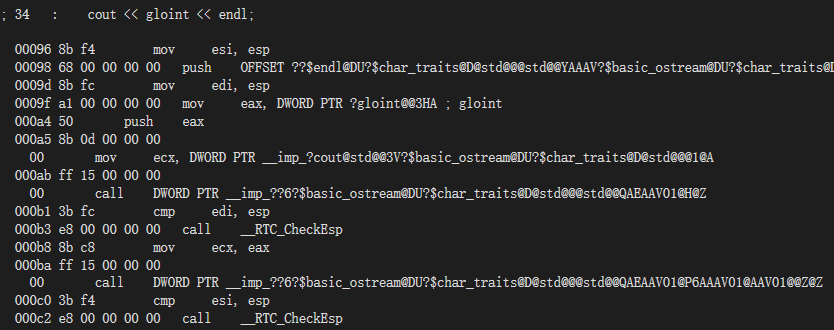
我们首先看函数参数和局部变量。我们知道，函数使用堆栈来传递参数，局部变量也是保存在堆栈上的。在函数体的开头对堆栈空间进行了初始化，编译器自动为函数预留了204字节的存储空间用来建立局部变量。我们可以看到形参a，b的偏移值都是正数，而局部变量c的偏移值是负数。所以，在这个堆栈框架中，我们就可以保存形参和局部变量了。 同时，我们也可以看到，函数将返回值保存在eax中，利用eax来返回。

接下来，我们来看函数的调用过程。第一步操作，是将实参按从右到左的顺序压栈；第二步，call指令完成两个动作，一个是将返回地址压入堆栈，二是将被调用过程的地址拷贝到指令指针寄存器eip中，CPU执行的下一条指令就是eip中保存的被调用的函数；函数执行完函数体，恢复寄存器的值，然后ret指令将返回地址的值从堆栈中弹出，送入指令指针寄存器eip中，是CPU返回到调用过程的地方继续执行，返回值保存了eax中，供调用程序使用。

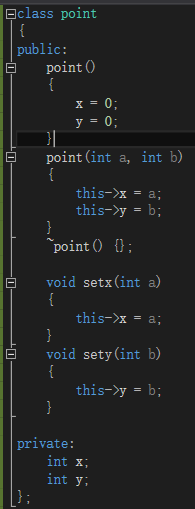
1. 全局变量

全局变量定义在了程序的最开头，且在变量名后面加了一些用于标识的后缀，定义完后，就对变量进行了赋值。全局变量是直接定义在内存里的，其生命周期是整个程序，所以任何子程序都可以直接使用全局变量。我们可以看main函数和max函数，其中定义的局部变量都是在堆栈上的，其生命周期是子程序，子程序结束，局部变量就会消亡。我们可以看一下全局变量的使用，与局部变量不同，全局变量的调用是直接使用变量名，而不是“寄存器+偏移量”这种寄存器变址寻址方式。

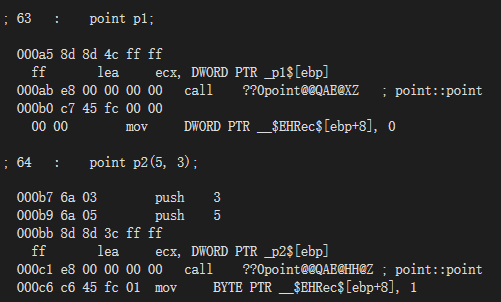
如果局部变量的变量名和全局变量的变量名重名，那么局部变量会将全局变量覆盖。



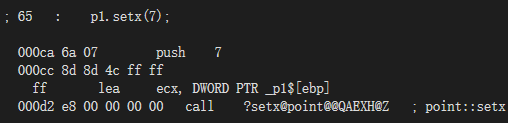
1. 类，成员函数，this指针，函数重载

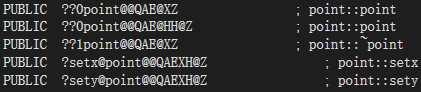
我们使用C++建立一个point类，如右图，为了方便，我们将成员函数的定义写在了类内。

我们可以看到了类的构造函数有两个，一个是没有任何参数的构造函数，另一个是带有两个int型形参的构造函数。这个两个函数同名，依靠形参的类型和个数的不同来区分，这就是C++中的函数重载。在汇编语言层面，我么可以看到，它们的名字后面加了不同的后缀，那么在调用的时候，就可以使用这两个带有不同后缀的函数名来区分重载的函数了。我们可以看p1和p2这两个使用不同的构造函数建立的类的实例，调用了具有不同后缀名的构造函数。



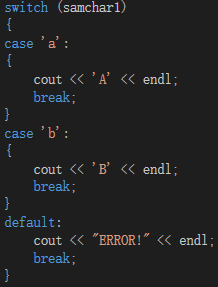
在类的外部重载的函数，也是通过不同的后缀名来区别的，这些后缀名都是编译器按照一定的规则，自动添加的。

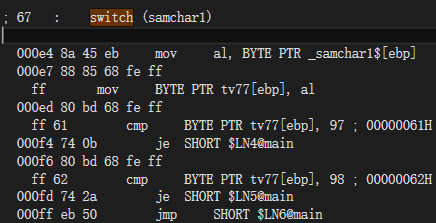
而类中的this指针，实际上就是函数中的一个局部变量。在调用成员函数时，把这个类的对象的地址作为一个形参，隐含地在函数调用时传给被调函数来使用。这个对象的地址不是通过堆栈来传递的，而是通过寄存器ecx来传递的。这一行代码是从成员函数sety中截取出来的，我们可以看到，将ecx的值赋给了this指针，而标号\_this$的值是-8，是一个负数，上面我们已经分析过，局部变量的这个标号值是负数，所以，我们可以得出结论，this指针的本质就是类的成员函数中的局部变量。

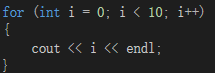
类的对象又是怎么去寻找它的成员函数呢？

我们看这段声明可以发现，除了构造函数和析构函数以外，成员函数的名字后都加了@point这样一个后缀，而构造函数和析构函数本身就和类名相同，所以，通过后缀名，就可以找到相应的成员函数了。

1. switch语句

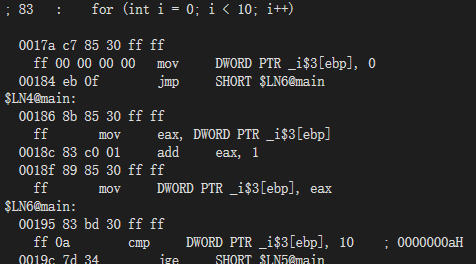
我们使用C++写了这样一个switch语句。

汇编代码中，首先将变量samchar1与a，b做比较，根据比较结果，跳转到相应的语句标号执行。所以，switch语句和if……else……语句没有本质的区别，都是先进行判断，然后执行相应的指令。



1. for循环

C++实现for语句如右图。

for语句的汇编代码如右图。For语句的执行，首先是使用cmp语句对循环条件做判断，然后执行循环体，执行一次循环体结束后，更新循环变量，再次判断循环条件，如此重复，直到循环条件不满足，跳出循环。这语C++的for循环的执行思路在本质上是相同的。

1. C++与汇编语言的优缺点及使用场合

C++等高级语言，和自然语言比较接近，可读性更好，编写程序时的效率更高。在编写大量代码，开发大型软件时，使用高级语言C++等更容易组织和维护，而且因为C++不会直接访问硬件，所以其平台移植性更好，不会受到硬件的限制。C++的很多特性，比如类，继承和派生，运算符重载等，非常适合大型软件的开发。但是C++因为代码量大，不能直接访问硬件，所以执行效率比较低，另外，因为语法规则的限制，某种程度上也限制了其使用。

而汇编语言是非常接近机器语言的，所以其可读性较差，编写程序的效率远不如高级语言C++等，而且由于汇编语言和硬件平台是直接相关的，所以对于不同的处理器，都需要使用不同的汇编语法，增加了开发难度。但是汇编语言可以直接访问硬件，速度快，代码的执行效率更高，在一些对速度要求比较高的场合，汇编语言的作用非常明显。

C++和汇编孰优孰劣很难下定论，只能说二者各有优劣，在不同的应用背景下，灵活地选择。C++可以调用汇编程序，汇编程序也可以调用使用C++开发的模块，这在很大程度上方便了我们的使用。比如我们开发一个大型项目，整体框架使用高级语言来开发，需要大量计算的部分，对速度要求很高的部分我们使用汇编来实现。

1. 其他收获

通过学习会汇编语言，我们不仅仅掌握了其语法，而且掌握和计算机相关的很多知识，对计算机的硬件底层有了更进一步的理解。通过编写汇编程序，我们有知道了C++层面的一些语法在汇编层面是怎么实现的，不仅加深了我们对C++的理解，也使我们在编写汇编语言的过程中在使用这些语法是犯的错误更少。在调试程序方面，我们也学到了如何在机器语言层面调试程序，可以查看反汇编代码，在汇编语言层面执行来寻找问题，查看寄存器的值，我们在了解了变量在内存中的存储方式以后，就可以观察内存窗口寻找问题。所以，汇编语言在学习计算机专业课的过程对我们是很有帮助的。

1. 附件1 C++源程序

main.cpp

#define \_CRL\_SECURE\_WARNINGS\_NO\_

#include <iostream>

#include <stdio.h>

#include <string>

using namespace std;

typedef struct samstruct

{

int samid;

char samname;

int samage;

}person;

class point

{

public:

point()

{

x = 0;

y = 0;

}

point(int a, int b)

{

this->x = a;

this->y = b;

}

~point() {};

void setx(int a)

{

this->x = a;

}

void sety(int b)

{

this->y = b;

}

private:

int x;

int y;

};

int max(int a, int b);

int gloint = 16;

int main()

{

char samchar1 = 'f';

int samint1 = 100;

int samint2 = -10;

short samshort1 = 10;

short sanmsht2 = -100;

unsigned int samUint1 = 10;

unsigned short samUshtint1 = 5;

int\* samptr1 = &samint1;

int & samint3 = samint2;

int samint4;

int gloint = 32;

person samper1 = { 1,'a',20 };

point p1;

point p2(5, 3);

p1.setx(7);

p1.sety(4);

switch (samchar1)

{

case 'a':

{

cout << 'A' << endl;

break;

}

case 'b':

{

cout << 'B' << endl;

break;

}

default:

cout << "ERROR!" << endl;

break;

}

for (int i = 0; i < 10; i++)

{

cout << i << endl;

}

samint4 = max(samint1, samint2);

cout << gloint << endl;

cout << samint4 << endl;

return 0;

}

int max(int a, int b)

{

int c = 0;

if (a > b)

{

c = a;

}

else

{

c = b;

}

return c;

}

1. 反汇编程序(注：附录只有反汇编分析中用到的部分，详细内容见附件)

main.cod

; Listing generated by Microsoft (R) Optimizing Compiler Version 19.00.23918.0

TITLE D:\ljb\课程\2016秋\汇编语言\大作业\Project1\Project1\main.cpp

.686P

.XMM

include listing.inc

.model flat

INCLUDELIB MSVCRTD

INCLUDELIB OLDNAMES

PUBLIC ?gloint@@3HA ; gloint

\_DATA SEGMENT

?gloint@@3HA DD 010H ; gloint

\_DATA ENDS

PUBLIC ?\_\_empty\_global\_delete@@YAXPAX@Z ; \_\_empty\_global\_delete

PUBLIC ?\_\_empty\_global\_delete@@YAXPAXI@Z ; \_\_empty\_global\_delete

PUBLIC ?length@?$char\_traits@D@std@@SAIPBD@Z ; std::char\_traits<char>::length

PUBLIC ?eq\_int\_type@?$char\_traits@D@std@@SA\_NABH0@Z ; std::char\_traits<char>::eq\_int\_type

PUBLIC ?eof@?$char\_traits@D@std@@SAHXZ ; std::char\_traits<char>::eof

PUBLIC ??0point@@QAE@XZ ; point::point

PUBLIC ??0point@@QAE@HH@Z ; point::point

PUBLIC ??1point@@QAE@XZ ; point::~point

PUBLIC ?setx@point@@QAEXH@Z ; point::setx

PUBLIC ?sety@point@@QAEXH@Z ; point::sety

PUBLIC ?max@@YAHHH@Z ; max

PUBLIC \_main

; Function compile flags: /Odtp /RTCsu /ZI

; File d:\ljb\课程\2016秋\汇编语言\大作业\project1\project1\main.cpp

; COMDAT \_main

\_TEXT SEGMENT

tv77 = -420 ; size = 4

$T2 = -412 ; size = 4

\_i$3 = -208 ; size = 4

\_p2$ = -196 ; size = 8

\_p1$ = -180 ; size = 8

\_samper1$ = -164 ; size = 12

\_gloint$ = -144 ; size = 4

\_samint4$ = -132 ; size = 4

\_samint3$ = -120 ; size = 4

\_samptr1$ = -108 ; size = 4

\_samUshtint1$ = -96 ; size = 2

\_samUint1$ = -84 ; size = 4

\_sanmsht2$ = -72 ; size = 2

\_samshort1$ = -60 ; size = 2

\_samint2$ = -48 ; size = 4

\_samint1$ = -36 ; size = 4

\_samchar1$ = -21 ; size = 1

\_\_$ArrayPad$ = -16 ; size = 4

\_\_$EHRec$ = -12 ; size = 12

\_main PROC ; COMDAT

; 50 : {

00000 55 push ebp

00001 8b ec mov ebp, esp

00003 6a ff push -1

00005 68 00 00 00 00 push \_\_ehhandler$\_main

0000a 64 a1 00 00 00

00 mov eax, DWORD PTR fs:0

00010 50 push eax

00011 81 ec 98 01 00

00 sub esp, 408 ; 00000198H

00017 53 push ebx

00018 56 push esi

00019 57 push edi

0001a 8d bd 5c fe ff

ff lea edi, DWORD PTR [ebp-420]

00020 b9 66 00 00 00 mov ecx, 102 ; 00000066H

00025 b8 cc cc cc cc mov eax, -858993460 ; ccccccccH

0002a f3 ab rep stosd

0002c a1 00 00 00 00 mov eax, DWORD PTR \_\_\_security\_cookie

00031 33 c5 xor eax, ebp

00033 89 45 f0 mov DWORD PTR \_\_$ArrayPad$[ebp], eax

00036 50 push eax

00037 8d 45 f4 lea eax, DWORD PTR \_\_$EHRec$[ebp]

0003a 64 a3 00 00 00

00 mov DWORD PTR fs:0, eax

; 51 : char samchar1 = 'f';

00040 c6 45 eb 66 mov BYTE PTR \_samchar1$[ebp], 102 ; 00000066H

; 52 : int samint1 = 100;

00044 c7 45 dc 64 00

00 00 mov DWORD PTR \_samint1$[ebp], 100 ; 00000064H

; 53 : int samint2 = -10;

0004b c7 45 d0 f6 ff

ff ff mov DWORD PTR \_samint2$[ebp], -10 ; fffffff6H

; 54 : short samshort1 = 10;

00052 b8 0a 00 00 00 mov eax, 10 ; 0000000aH

00057 66 89 45 c4 mov WORD PTR \_samshort1$[ebp], ax

; 55 : short sanmsht2 = -100;

0005b b8 9c ff ff ff mov eax, -100 ; ffffff9cH

00060 66 89 45 b8 mov WORD PTR \_sanmsht2$[ebp], ax

; 56 : unsigned int samUint1 = 10;

00064 c7 45 ac 0a 00

00 00 mov DWORD PTR \_samUint1$[ebp], 10 ; 0000000aH

; 57 : unsigned short samUshtint1 = 5;

0006b b8 05 00 00 00 mov eax, 5

00070 66 89 45 a0 mov WORD PTR \_samUshtint1$[ebp], ax

; 58 : int\* samptr1 = &samint1;

00074 8d 45 dc lea eax, DWORD PTR \_samint1$[ebp]

00077 89 45 94 mov DWORD PTR \_samptr1$[ebp], eax

; 59 : int & samint3 = samint2;

0007a 8d 45 d0 lea eax, DWORD PTR \_samint2$[ebp]

0007d 89 45 88 mov DWORD PTR \_samint3$[ebp], eax

; 60 : int samint4;

; 61 : int gloint = 32;

00080 c7 85 70 ff ff

ff 20 00 00 00 mov DWORD PTR \_gloint$[ebp], 32 ; 00000020H

; 62 : person samper1 = { 1,'a',20 };

0008a c7 85 5c ff ff

ff 01 00 00 00 mov DWORD PTR \_samper1$[ebp], 1

00094 c6 85 60 ff ff

ff 61 mov BYTE PTR \_samper1$[ebp+4], 97 ; 00000061H

0009b c7 85 64 ff ff

ff 14 00 00 00 mov DWORD PTR \_samper1$[ebp+8], 20 ; 00000014H

; 63 : point p1;

000a5 8d 8d 4c ff ff

ff lea ecx, DWORD PTR \_p1$[ebp]

000ab e8 00 00 00 00 call ??0point@@QAE@XZ ; point::point

000b0 c7 45 fc 00 00

00 00 mov DWORD PTR \_\_$EHRec$[ebp+8], 0

; 64 : point p2(5, 3);

000b7 6a 03 push 3

000b9 6a 05 push 5

000bb 8d 8d 3c ff ff

ff lea ecx, DWORD PTR \_p2$[ebp]

000c1 e8 00 00 00 00 call ??0point@@QAE@HH@Z ; point::point

000c6 c6 45 fc 01 mov BYTE PTR \_\_$EHRec$[ebp+8], 1

; 65 : p1.setx(7);

000ca 6a 07 push 7

000cc 8d 8d 4c ff ff

ff lea ecx, DWORD PTR \_p1$[ebp]

000d2 e8 00 00 00 00 call ?setx@point@@QAEXH@Z ; point::setx

; 66 : p1.sety(4);

000d7 6a 04 push 4

000d9 8d 8d 4c ff ff

ff lea ecx, DWORD PTR \_p1$[ebp]

000df e8 00 00 00 00 call ?sety@point@@QAEXH@Z ; point::sety

; 67 : switch (samchar1)

000e4 8a 45 eb mov al, BYTE PTR \_samchar1$[ebp]

000e7 88 85 5c fe ff

ff mov BYTE PTR tv77[ebp], al

000ed 80 bd 5c fe ff

ff 61 cmp BYTE PTR tv77[ebp], 97 ; 00000061H

000f4 74 0b je SHORT $LN7@main

000f6 80 bd 5c fe ff

ff 62 cmp BYTE PTR tv77[ebp], 98 ; 00000062H

000fd 74 2a je SHORT $LN8@main

000ff eb 50 jmp SHORT $LN9@main

$LN7@main:

; 68 : {

; 69 : case 'a':

; 70 : {

; 71 : cout << 'A' << endl;

00101 8b f4 mov esi, esp

00103 68 00 00 00 00 push OFFSET ??$endl@DU?$char\_traits@D@std@@@std@@YAAAV?$basic\_ostream@DU?$char\_traits@D@std@@@0@AAV10@@Z ; std::endl<char,std::char\_traits<char> >

00108 6a 41 push 65 ; 00000041H

0010a a1 00 00 00 00 mov eax, DWORD PTR \_\_imp\_?cout@std@@3V?$basic\_ostream@DU?$char\_traits@D@std@@@1@A

0010f 50 push eax

00110 e8 00 00 00 00 call ??$?6U?$char\_traits@D@std@@@std@@YAAAV?$basic\_ostream@DU?$char\_traits@D@std@@@0@AAV10@D@Z ; std::operator<<<std::char\_traits<char> >

00115 83 c4 08 add esp, 8

00118 8b c8 mov ecx, eax

0011a ff 15 00 00 00

00 call DWORD PTR \_\_imp\_??6?$basic\_ostream@DU?$char\_traits@D@std@@@std@@QAEAAV01@P6AAAV01@AAV01@@Z@Z

00120 3b f4 cmp esi, esp

00122 e8 00 00 00 00 call \_\_RTC\_CheckEsp

; 72 : break;

00127 eb 51 jmp SHORT $LN2@main

$LN8@main:

; 73 : }

; 74 : case 'b':

; 75 : {

; 76 : cout << 'B' << endl;

00129 8b f4 mov esi, esp

0012b 68 00 00 00 00 push OFFSET ??$endl@DU?$char\_traits@D@std@@@std@@YAAAV?$basic\_ostream@DU?$char\_traits@D@std@@@0@AAV10@@Z ; std::endl<char,std::char\_traits<char> >

00130 6a 42 push 66 ; 00000042H

00132 a1 00 00 00 00 mov eax, DWORD PTR \_\_imp\_?cout@std@@3V?$basic\_ostream@DU?$char\_traits@D@std@@@1@A

00137 50 push eax

00138 e8 00 00 00 00 call ??$?6U?$char\_traits@D@std@@@std@@YAAAV?$basic\_ostream@DU?$char\_traits@D@std@@@0@AAV10@D@Z ; std::operator<<<std::char\_traits<char> >

0013d 83 c4 08 add esp, 8

00140 8b c8 mov ecx, eax

00142 ff 15 00 00 00

00 call DWORD PTR \_\_imp\_??6?$basic\_ostream@DU?$char\_traits@D@std@@@std@@QAEAAV01@P6AAAV01@AAV01@@Z@Z

00148 3b f4 cmp esi, esp

0014a e8 00 00 00 00 call \_\_RTC\_CheckEsp

; 77 : break;

0014f eb 29 jmp SHORT $LN2@main

$LN9@main:

; 78 : }

; 79 : default:

; 80 : cout << "ERROR!" << endl;

00151 8b f4 mov esi, esp

00153 68 00 00 00 00 push OFFSET ??$endl@DU?$char\_traits@D@std@@@std@@YAAAV?$basic\_ostream@DU?$char\_traits@D@std@@@0@AAV10@@Z ; std::endl<char,std::char\_traits<char> >

00158 68 00 00 00 00 push OFFSET ??\_C@\_06FHOFOIFO@ERROR?$CB?$AA@

0015d a1 00 00 00 00 mov eax, DWORD PTR \_\_imp\_?cout@std@@3V?$basic\_ostream@DU?$char\_traits@D@std@@@1@A

00162 50 push eax

00163 e8 00 00 00 00 call ??$?6U?$char\_traits@D@std@@@std@@YAAAV?$basic\_ostream@DU?$char\_traits@D@std@@@0@AAV10@PBD@Z ; std::operator<<<std::char\_traits<char> >

00168 83 c4 08 add esp, 8

0016b 8b c8 mov ecx, eax

0016d ff 15 00 00 00

00 call DWORD PTR \_\_imp\_??6?$basic\_ostream@DU?$char\_traits@D@std@@@std@@QAEAAV01@P6AAAV01@AAV01@@Z@Z

00173 3b f4 cmp esi, esp

00175 e8 00 00 00 00 call \_\_RTC\_CheckEsp

$LN2@main:

; 81 : break;

; 82 : }

; 83 : for (int i = 0; i < 10; i++)

0017a c7 85 30 ff ff

ff 00 00 00 00 mov DWORD PTR \_i$3[ebp], 0

00184 eb 0f jmp SHORT $LN6@main

$LN4@main:

00186 8b 85 30 ff ff

ff mov eax, DWORD PTR \_i$3[ebp]

0018c 83 c0 01 add eax, 1

0018f 89 85 30 ff ff

ff mov DWORD PTR \_i$3[ebp], eax

$LN6@main:

00195 83 bd 30 ff ff

ff 0a cmp DWORD PTR \_i$3[ebp], 10 ; 0000000aH

0019c 7d 34 jge SHORT $LN5@main

; 84 : {

; 85 : cout << i << endl;

0019e 8b f4 mov esi, esp

001a0 68 00 00 00 00 push OFFSET ??$endl@DU?$char\_traits@D@std@@@std@@YAAAV?$basic\_ostream@DU?$char\_traits@D@std@@@0@AAV10@@Z ; std::endl<char,std::char\_traits<char> >

001a5 8b fc mov edi, esp

001a7 8b 85 30 ff ff

ff mov eax, DWORD PTR \_i$3[ebp]

001ad 50 push eax

001ae 8b 0d 00 00 00

00 mov ecx, DWORD PTR \_\_imp\_?cout@std@@3V?$basic\_ostream@DU?$char\_traits@D@std@@@1@A

001b4 ff 15 00 00 00

00 call DWORD PTR \_\_imp\_??6?$basic\_ostream@DU?$char\_traits@D@std@@@std@@QAEAAV01@H@Z

001ba 3b fc cmp edi, esp

001bc e8 00 00 00 00 call \_\_RTC\_CheckEsp

001c1 8b c8 mov ecx, eax

001c3 ff 15 00 00 00

00 call DWORD PTR \_\_imp\_??6?$basic\_ostream@DU?$char\_traits@D@std@@@std@@QAEAAV01@P6AAAV01@AAV01@@Z@Z

001c9 3b f4 cmp esi, esp

001cb e8 00 00 00 00 call \_\_RTC\_CheckEsp

; 86 : }

001d0 eb b4 jmp SHORT $LN4@main

$LN5@main:

; 87 : samint4 = max(samint1, samint2);

001d2 8b 45 d0 mov eax, DWORD PTR \_samint2$[ebp]

001d5 50 push eax

001d6 8b 4d dc mov ecx, DWORD PTR \_samint1$[ebp]

001d9 51 push ecx

001da e8 00 00 00 00 call ?max@@YAHHH@Z ; max

001df 83 c4 08 add esp, 8

001e2 89 85 7c ff ff

ff mov DWORD PTR \_samint4$[ebp], eax

; 88 : cout << gloint << endl;

001e8 8b f4 mov esi, esp

001ea 68 00 00 00 00 push OFFSET ??$endl@DU?$char\_traits@D@std@@@std@@YAAAV?$basic\_ostream@DU?$char\_traits@D@std@@@0@AAV10@@Z ; std::endl<char,std::char\_traits<char> >

001ef 8b fc mov edi, esp

001f1 8b 85 70 ff ff

ff mov eax, DWORD PTR \_gloint$[ebp]

001f7 50 push eax

001f8 8b 0d 00 00 00

00 mov ecx, DWORD PTR \_\_imp\_?cout@std@@3V?$basic\_ostream@DU?$char\_traits@D@std@@@1@A

001fe ff 15 00 00 00

00 call DWORD PTR \_\_imp\_??6?$basic\_ostream@DU?$char\_traits@D@std@@@std@@QAEAAV01@H@Z

00204 3b fc cmp edi, esp

00206 e8 00 00 00 00 call \_\_RTC\_CheckEsp

0020b 8b c8 mov ecx, eax

0020d ff 15 00 00 00

00 call DWORD PTR \_\_imp\_??6?$basic\_ostream@DU?$char\_traits@D@std@@@std@@QAEAAV01@P6AAAV01@AAV01@@Z@Z

00213 3b f4 cmp esi, esp

00215 e8 00 00 00 00 call \_\_RTC\_CheckEsp

; 89 : cout << samint4 << endl;

0021a 8b f4 mov esi, esp

0021c 68 00 00 00 00 push OFFSET ??$endl@DU?$char\_traits@D@std@@@std@@YAAAV?$basic\_ostream@DU?$char\_traits@D@std@@@0@AAV10@@Z ; std::endl<char,std::char\_traits<char> >

00221 8b fc mov edi, esp

00223 8b 85 7c ff ff

ff mov eax, DWORD PTR \_samint4$[ebp]

00229 50 push eax

0022a 8b 0d 00 00 00

00 mov ecx, DWORD PTR \_\_imp\_?cout@std@@3V?$basic\_ostream@DU?$char\_traits@D@std@@@1@A

00230 ff 15 00 00 00

00 call DWORD PTR \_\_imp\_??6?$basic\_ostream@DU?$char\_traits@D@std@@@std@@QAEAAV01@H@Z

00236 3b fc cmp edi, esp

00238 e8 00 00 00 00 call \_\_RTC\_CheckEsp

0023d 8b c8 mov ecx, eax

0023f ff 15 00 00 00

00 call DWORD PTR \_\_imp\_??6?$basic\_ostream@DU?$char\_traits@D@std@@@std@@QAEAAV01@P6AAAV01@AAV01@@Z@Z

00245 3b f4 cmp esi, esp

00247 e8 00 00 00 00 call \_\_RTC\_CheckEsp

; 90 : return 0;

0024c c7 85 64 fe ff

ff 00 00 00 00 mov DWORD PTR $T2[ebp], 0

00256 c6 45 fc 00 mov BYTE PTR \_\_$EHRec$[ebp+8], 0

0025a 8d 8d 3c ff ff

ff lea ecx, DWORD PTR \_p2$[ebp]

00260 e8 00 00 00 00 call ??1point@@QAE@XZ ; point::~point

00265 c7 45 fc ff ff

ff ff mov DWORD PTR \_\_$EHRec$[ebp+8], -1

0026c 8d 8d 4c ff ff

ff lea ecx, DWORD PTR \_p1$[ebp]

00272 e8 00 00 00 00 call ??1point@@QAE@XZ ; point::~point

00277 8b 85 64 fe ff

ff mov eax, DWORD PTR $T2[ebp]

; 91 : }

; Function compile flags: /Odtp /RTCsu /ZI

; File d:\ljb\课程\2016秋\汇编语言\大作业\project1\project1\main.cpp

; COMDAT ?max@@YAHHH@Z

\_TEXT SEGMENT

\_c$ = -8 ; size = 4

\_a$ = 8 ; size = 4

\_b$ = 12 ; size = 4

?max@@YAHHH@Z PROC ; max, COMDAT

; 94 : {

00000 55 push ebp

00001 8b ec mov ebp, esp

00003 81 ec cc 00 00

00 sub esp, 204 ; 000000ccH

00009 53 push ebx

0000a 56 push esi

0000b 57 push edi

0000c 8d bd 34 ff ff

ff lea edi, DWORD PTR [ebp-204]

00012 b9 33 00 00 00 mov ecx, 51 ; 00000033H

00017 b8 cc cc cc cc mov eax, -858993460 ; ccccccccH

0001c f3 ab rep stosd

; 95 : int c = 0;

0001e c7 45 f8 00 00

00 00 mov DWORD PTR \_c$[ebp], 0

; 96 : if (a > b)

00025 8b 45 08 mov eax, DWORD PTR \_a$[ebp]

00028 3b 45 0c cmp eax, DWORD PTR \_b$[ebp]

0002b 7e 08 jle SHORT $LN2@max

; 97 : {

; 98 : c = a;

0002d 8b 45 08 mov eax, DWORD PTR \_a$[ebp]

00030 89 45 f8 mov DWORD PTR \_c$[ebp], eax

; 99 : }

; 100 : else

00033 eb 06 jmp SHORT $LN3@max

$LN2@max:

; 101 : {

; 102 : c = b;

00035 8b 45 0c mov eax, DWORD PTR \_b$[ebp]

00038 89 45 f8 mov DWORD PTR \_c$[ebp], eax

$LN3@max:

; 103 : }

; 104 : return c;

0003b 8b 45 f8 mov eax, DWORD PTR \_c$[ebp]

; 105 : }

0003e 5f pop edi

0003f 5e pop esi

00040 5b pop ebx

00041 8b e5 mov esp, ebp

00043 5d pop ebp

00044 c3 ret 0

?max@@YAHHH@Z ENDP ; max

\_TEXT ENDS

; Function compile flags: /Odtp /RTCsu /ZI

; File d:\ljb\课程\2016秋\汇编语言\大作业\project1\project1\main.cpp

; COMDAT ?sety@point@@QAEXH@Z

\_TEXT SEGMENT

\_this$ = -8 ; size = 4

\_b$ = 8 ; size = 4

?sety@point@@QAEXH@Z PROC ; point::sety, COMDAT

; \_this$ = ecx

; 36 : {

00000 55 push ebp

00001 8b ec mov ebp, esp

00003 81 ec cc 00 00

00 sub esp, 204 ; 000000ccH

00009 53 push ebx

0000a 56 push esi

0000b 57 push edi

0000c 51 push ecx

0000d 8d bd 34 ff ff

ff lea edi, DWORD PTR [ebp-204]

00013 b9 33 00 00 00 mov ecx, 51 ; 00000033H

00018 b8 cc cc cc cc mov eax, -858993460 ; ccccccccH

0001d f3 ab rep stosd

0001f 59 pop ecx

00020 89 4d f8 mov DWORD PTR \_this$[ebp], ecx

; 37 : this->y = b;

00023 8b 45 f8 mov eax, DWORD PTR \_this$[ebp]

00026 8b 4d 08 mov ecx, DWORD PTR \_b$[ebp]

00029 89 48 04 mov DWORD PTR [eax+4], ecx

; 38 : }

0002c 5f pop edi

0002d 5e pop esi

0002e 5b pop ebx

0002f 8b e5 mov esp, ebp

00031 5d pop ebp

00032 c2 04 00 ret 4

?sety@point@@QAEXH@Z ENDP ; point::sety

\_TEXT ENDS

; Function compile flags: /Odtp /RTCsu /ZI

; File d:\ljb\课程\2016秋\汇编语言\大作业\project1\project1\main.cpp

; COMDAT ?setx@point@@QAEXH@Z

\_TEXT SEGMENT

\_this$ = -8 ; size = 4

\_a$ = 8 ; size = 4

?setx@point@@QAEXH@Z PROC ; point::setx, COMDAT

; \_this$ = ecx

; 32 : {

00000 55 push ebp

00001 8b ec mov ebp, esp

00003 81 ec cc 00 00

00 sub esp, 204 ; 000000ccH

00009 53 push ebx

0000a 56 push esi

0000b 57 push edi

0000c 51 push ecx

0000d 8d bd 34 ff ff

ff lea edi, DWORD PTR [ebp-204]

00013 b9 33 00 00 00 mov ecx, 51 ; 00000033H

00018 b8 cc cc cc cc mov eax, -858993460 ; ccccccccH

0001d f3 ab rep stosd

0001f 59 pop ecx

00020 89 4d f8 mov DWORD PTR \_this$[ebp], ecx

; 33 : this->x = a;

00023 8b 45 f8 mov eax, DWORD PTR \_this$[ebp]

00026 8b 4d 08 mov ecx, DWORD PTR \_a$[ebp]

00029 89 08 mov DWORD PTR [eax], ecx

; 34 : }

0002b 5f pop edi

0002c 5e pop esi

0002d 5b pop ebx

0002e 8b e5 mov esp, ebp

00030 5d pop ebp

00031 c2 04 00 ret 4

?setx@point@@QAEXH@Z ENDP ; point::setx

\_TEXT ENDS

; Function compile flags: /Odtp /RTCsu /ZI

; File d:\ljb\课程\2016秋\汇编语言\大作业\project1\project1\main.cpp

; COMDAT ??1point@@QAE@XZ

\_TEXT SEGMENT

\_this$ = -8 ; size = 4

??1point@@QAE@XZ PROC ; point::~point, COMDAT

; \_this$ = ecx

; 29 : ~point() {};

00000 55 push ebp

00001 8b ec mov ebp, esp

00003 81 ec cc 00 00

00 sub esp, 204 ; 000000ccH

00009 53 push ebx

0000a 56 push esi

0000b 57 push edi

0000c 51 push ecx

0000d 8d bd 34 ff ff

ff lea edi, DWORD PTR [ebp-204]

00013 b9 33 00 00 00 mov ecx, 51 ; 00000033H

00018 b8 cc cc cc cc mov eax, -858993460 ; ccccccccH

0001d f3 ab rep stosd

0001f 59 pop ecx

00020 89 4d f8 mov DWORD PTR \_this$[ebp], ecx

00023 5f pop edi

00024 5e pop esi

00025 5b pop ebx

00026 8b e5 mov esp, ebp

00028 5d pop ebp

00029 c3 ret 0

??1point@@QAE@XZ ENDP ; point::~point

\_TEXT ENDS

; Function compile flags: /Odtp /RTCsu /ZI

; File d:\ljb\课程\2016秋\汇编语言\大作业\project1\project1\main.cpp

; COMDAT ??0point@@QAE@HH@Z

\_TEXT SEGMENT

\_this$ = -8 ; size = 4

\_a$ = 8 ; size = 4

\_b$ = 12 ; size = 4

??0point@@QAE@HH@Z PROC ; point::point, COMDAT

; \_this$ = ecx

; 24 : point(int a, int b)

00000 55 push ebp

00001 8b ec mov ebp, esp

00003 81 ec cc 00 00

00 sub esp, 204 ; 000000ccH

00009 53 push ebx

0000a 56 push esi

0000b 57 push edi

0000c 51 push ecx

0000d 8d bd 34 ff ff

ff lea edi, DWORD PTR [ebp-204]

00013 b9 33 00 00 00 mov ecx, 51 ; 00000033H

00018 b8 cc cc cc cc mov eax, -858993460 ; ccccccccH

0001d f3 ab rep stosd

0001f 59 pop ecx

00020 89 4d f8 mov DWORD PTR \_this$[ebp], ecx

; 25 : {

; 26 : this->x = a;

00023 8b 45 f8 mov eax, DWORD PTR \_this$[ebp]

00026 8b 4d 08 mov ecx, DWORD PTR \_a$[ebp]

00029 89 08 mov DWORD PTR [eax], ecx

; 27 : this->y = b;

0002b 8b 45 f8 mov eax, DWORD PTR \_this$[ebp]

0002e 8b 4d 0c mov ecx, DWORD PTR \_b$[ebp]

00031 89 48 04 mov DWORD PTR [eax+4], ecx

; 28 : }

00034 8b 45 f8 mov eax, DWORD PTR \_this$[ebp]

00037 5f pop edi

00038 5e pop esi

00039 5b pop ebx

0003a 8b e5 mov esp, ebp

0003c 5d pop ebp

0003d c2 08 00 ret 8

??0point@@QAE@HH@Z ENDP ; point::point

\_TEXT ENDS

; Function compile flags: /Odtp /RTCsu /ZI

; File d:\ljb\课程\2016秋\汇编语言\大作业\project1\project1\main.cpp

; COMDAT ??0point@@QAE@XZ

\_TEXT SEGMENT

\_this$ = -8 ; size = 4

??0point@@QAE@XZ PROC ; point::point, COMDAT

; \_this$ = ecx

; 19 : point()

00000 55 push ebp

00001 8b ec mov ebp, esp

00003 81 ec cc 00 00

00 sub esp, 204 ; 000000ccH

00009 53 push ebx

0000a 56 push esi

0000b 57 push edi

0000c 51 push ecx

0000d 8d bd 34 ff ff

ff lea edi, DWORD PTR [ebp-204]

00013 b9 33 00 00 00 mov ecx, 51 ; 00000033H

00018 b8 cc cc cc cc mov eax, -858993460 ; ccccccccH

0001d f3 ab rep stosd

0001f 59 pop ecx

00020 89 4d f8 mov DWORD PTR \_this$[ebp], ecx

; 20 : {

; 21 : x = 0;

00023 8b 45 f8 mov eax, DWORD PTR \_this$[ebp]

00026 c7 00 00 00 00

00 mov DWORD PTR [eax], 0

; 22 : y = 0;

0002c 8b 45 f8 mov eax, DWORD PTR \_this$[ebp]

0002f c7 40 04 00 00

00 00 mov DWORD PTR [eax+4], 0

; 23 : }

00036 8b 45 f8 mov eax, DWORD PTR \_this$[ebp]

00039 5f pop edi

0003a 5e pop esi

0003b 5b pop ebx

0003c 8b e5 mov esp, ebp

0003e 5d pop ebp

0003f c3 ret 0

??0point@@QAE@XZ ENDP ; point::point

\_TEXT ENDS

; Function compile flags: /Odtp /RTCsu /ZI

; File c:\program files (x86)\microsoft visual studio 14.0\vc\include\iosfwd

; COMDAT ?eof@?$char\_traits@D@std@@SAHXZ

\_TEXT SEGMENT

?eof@?$char\_traits@D@std@@SAHXZ PROC ; std::char\_traits<char>::eof, COMDAT

; 605 : { // return end-of-file metacharacter

00000 55 push ebp

00001 8b ec mov ebp, esp

00003 81 ec c0 00 00

00 sub esp, 192 ; 000000c0H

00009 53 push ebx

0000a 56 push esi

0000b 57 push edi

0000c 8d bd 40 ff ff

ff lea edi, DWORD PTR [ebp-192]

00012 b9 30 00 00 00 mov ecx, 48 ; 00000030H

00017 b8 cc cc cc cc mov eax, -858993460 ; ccccccccH

0001c f3 ab rep stosd

; 606 : return (EOF);

0001e 83 c8 ff or eax, -1

; 607 : }

00021 5f pop edi

00022 5e pop esi

00023 5b pop ebx

00024 8b e5 mov esp, ebp

00026 5d pop ebp

00027 c3 ret 0

?eof@?$char\_traits@D@std@@SAHXZ ENDP ; std::char\_traits<char>::eof

\_TEXT ENDS

; Function compile flags: /Odtp /RTCsu /ZI

; File c:\program files (x86)\microsoft visual studio 14.0\vc\include\iosfwd

; COMDAT ?eq\_int\_type@?$char\_traits@D@std@@SA\_NABH0@Z

\_TEXT SEGMENT

tv65 = -196 ; size = 4

\_\_Left$ = 8 ; size = 4

\_\_Right$ = 12 ; size = 4

?eq\_int\_type@?$char\_traits@D@std@@SA\_NABH0@Z PROC ; std::char\_traits<char>::eq\_int\_type, COMDAT

; 594 : { // test for metacharacter equality

00000 55 push ebp

00001 8b ec mov ebp, esp

00003 81 ec c4 00 00

00 sub esp, 196 ; 000000c4H

00009 53 push ebx

0000a 56 push esi

0000b 57 push edi

0000c 8d bd 3c ff ff

ff lea edi, DWORD PTR [ebp-196]

00012 b9 31 00 00 00 mov ecx, 49 ; 00000031H

00017 b8 cc cc cc cc mov eax, -858993460 ; ccccccccH

0001c f3 ab rep stosd

; 595 : return (\_Left == \_Right);

0001e 8b 45 08 mov eax, DWORD PTR \_\_Left$[ebp]

00021 8b 4d 0c mov ecx, DWORD PTR \_\_Right$[ebp]

00024 8b 10 mov edx, DWORD PTR [eax]

00026 3b 11 cmp edx, DWORD PTR [ecx]

00028 75 0c jne SHORT $LN3@eq\_int\_typ

0002a c7 85 3c ff ff

ff 01 00 00 00 mov DWORD PTR tv65[ebp], 1

00034 eb 0a jmp SHORT $LN4@eq\_int\_typ

$LN3@eq\_int\_typ:

00036 c7 85 3c ff ff

ff 00 00 00 00 mov DWORD PTR tv65[ebp], 0

$LN4@eq\_int\_typ:

00040 8a 85 3c ff ff

ff mov al, BYTE PTR tv65[ebp]

; 596 : }

00046 5f pop edi

00047 5e pop esi

00048 5b pop ebx

00049 8b e5 mov esp, ebp

0004b 5d pop ebp

0004c c3 ret 0

?eq\_int\_type@?$char\_traits@D@std@@SA\_NABH0@Z ENDP ; std::char\_traits<char>::eq\_int\_type

\_TEXT ENDS

; Function compile flags: /Odtp /RTCsu /ZI

; File c:\program files (x86)\microsoft visual studio 14.0\vc\include\iosfwd

; COMDAT ?length@?$char\_traits@D@std@@SAIPBD@Z

\_TEXT SEGMENT

tv68 = -196 ; size = 4

\_\_First$ = 8 ; size = 4

?length@?$char\_traits@D@std@@SAIPBD@Z PROC ; std::char\_traits<char>::length, COMDAT

; 522 : { // find length of null-terminated string

00000 55 push ebp

00001 8b ec mov ebp, esp

00003 81 ec c4 00 00

00 sub esp, 196 ; 000000c4H

00009 53 push ebx

0000a 56 push esi

0000b 57 push edi

0000c 8d bd 3c ff ff

ff lea edi, DWORD PTR [ebp-196]

00012 b9 31 00 00 00 mov ecx, 49 ; 00000031H

00017 b8 cc cc cc cc mov eax, -858993460 ; ccccccccH

0001c f3 ab rep stosd

; 523 : return (\*\_First == 0 ? 0

0001e 8b 45 08 mov eax, DWORD PTR \_\_First$[ebp]

00021 0f be 08 movsx ecx, BYTE PTR [eax]

00024 85 c9 test ecx, ecx

00026 75 0c jne SHORT $LN3@length

00028 c7 85 3c ff ff

ff 00 00 00 00 mov DWORD PTR tv68[ebp], 0

00032 eb 12 jmp SHORT $LN4@length

$LN3@length:

00034 8b 55 08 mov edx, DWORD PTR \_\_First$[ebp]

00037 52 push edx

00038 e8 00 00 00 00 call \_strlen

0003d 83 c4 04 add esp, 4

00040 89 85 3c ff ff

ff mov DWORD PTR tv68[ebp], eax

$LN4@length:

00046 8b 85 3c ff ff

ff mov eax, DWORD PTR tv68[ebp]

; 524 : : \_CSTD strlen(\_First));

; 525 : }

0004c 5f pop edi

0004d 5e pop esi

0004e 5b pop ebx

0004f 81 c4 c4 00 00

00 add esp, 196 ; 000000c4H

00055 3b ec cmp ebp, esp

00057 e8 00 00 00 00 call \_\_RTC\_CheckEsp

0005c 8b e5 mov esp, ebp

0005e 5d pop ebp

0005f c3 ret 0

?length@?$char\_traits@D@std@@SAIPBD@Z ENDP ; std::char\_traits<char>::length

\_TEXT ENDS

; Function compile flags: /Odtp /RTCsu /ZI

; File d:\ljb\课程\2016秋\汇编语言\大作业\project1\project1\main.cpp

; COMDAT ?\_\_empty\_global\_delete@@YAXPAXI@Z

\_TEXT SEGMENT

\_\_\_formal$ = 8 ; size = 4

\_\_\_formal$ = 12 ; size = 4

?\_\_empty\_global\_delete@@YAXPAXI@Z PROC ; \_\_empty\_global\_delete, COMDAT

; 106 : }

00000 55 push ebp

00001 8b ec mov ebp, esp

00003 81 ec c0 00 00

00 sub esp, 192 ; 000000c0H

00009 53 push ebx

0000a 56 push esi

0000b 57 push edi

0000c 8d bd 40 ff ff

ff lea edi, DWORD PTR [ebp-192]

00012 b9 30 00 00 00 mov ecx, 48 ; 00000030H

00017 b8 cc cc cc cc mov eax, -858993460 ; ccccccccH

0001c f3 ab rep stosd

0001e 5f pop edi

0001f 5e pop esi

00020 5b pop ebx

00021 8b e5 mov esp, ebp

00023 5d pop ebp

00024 c3 ret 0

?\_\_empty\_global\_delete@@YAXPAXI@Z ENDP ; \_\_empty\_global\_delete

\_TEXT ENDS

; Function compile flags: /Odtp /RTCsu /ZI

; File d:\ljb\课程\2016秋\汇编语言\大作业\project1\project1\main.cpp

; COMDAT ?\_\_empty\_global\_delete@@YAXPAX@Z

\_TEXT SEGMENT

\_\_\_formal$ = 8 ; size = 4

?\_\_empty\_global\_delete@@YAXPAX@Z PROC ; \_\_empty\_global\_delete, COMDAT

; 106 : }

00000 55 push ebp

00001 8b ec mov ebp, esp

00003 81 ec c0 00 00

00 sub esp, 192 ; 000000c0H

00009 53 push ebx

0000a 56 push esi

0000b 57 push edi

0000c 8d bd 40 ff ff

ff lea edi, DWORD PTR [ebp-192]

00012 b9 30 00 00 00 mov ecx, 48 ; 00000030H

00017 b8 cc cc cc cc mov eax, -858993460 ; ccccccccH

0001c f3 ab rep stosd

0001e 5f pop edi

0001f 5e pop esi

00020 5b pop ebx

00021 8b e5 mov esp, ebp

00023 5d pop ebp

00024 c3 ret 0

?\_\_empty\_global\_delete@@YAXPAX@Z ENDP ; \_\_empty\_global\_delete

\_TEXT ENDS

END