

**课 程 实 验 报 告**

**课程名称： 汇编语言程序设计实验**

**实验名称： 实验三 模块化程序设计**

**实验时间： 2018-4-9，14：00-17：30 实验地点： 南一楼804室**

**指导教师：**

**专业班级：计算机科学与技术201706班**

**学 号： U201714761 姓 名： 胡澳**

**同组学生： 西月栋 报告日期： 2018年 4 月 16 日**

**原创性声明**

本人郑重声明：本报告的内容由本人独立完成，有关观点、方法、数据和文献等的引用已经在文中指出。除文中已经注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品或成果，不存在剽窃、抄袭行为。

特此声明！

学生签名：

日期：2018.4.16

成绩评定

|  |  |  |
| --- | --- | --- |
| 实验完成质量得分（70分）（实验步骤清晰详细深入，实验记录真实完整等） | 报告撰写质量得分（30分）（报告规范、完整、通顺、详实等） | 总成绩（100分） |
|  |  |  |

指导教师签字：

日期：

目 录

[1 实验目的与要求 2](#_Toc6761914)

[2 实验内容 2](#_Toc6761915)

[3 实验过程 5](#_Toc6761916)

[3.1 任务1 5](#_Toc6761917)

[3.1.1 设计思路 5](#_Toc6761918)

[3.1.2 流程图 5](#_Toc6761919)

[3.1.3 源代码 6](#_Toc6761920)

[3.1.4 实验步骤 17](#_Toc6761921)

[3.1.5 实验记录和分析 17](#_Toc6761922)

[3.2 任务2 20](#_Toc6761923)

[3.2.1 设计思路 20](#_Toc6761924)

[3.2.2 流程图 20](#_Toc6761925)

[3.2.3 源代码 20](#_Toc6761926)

[3.2.4 实验步骤 26](#_Toc6761927)

[3.2.5 实验记录和分析 26](#_Toc6761928)

[4 总结和体会 29](#_Toc6761929)

# 1 实验目的与要求

(1) 掌握子程序设计的方法与技巧，熟悉子程序的参数传递方法和调用原理；

(2) 掌握宏指令、模块化程序的设计方法;

(3) 掌握较大规模程序的合作开发与调试方法；

(4) 掌握汇编语言程序与C语言程序混合编程的方法；

(5) 了解C编译器的基本优化方法;

(6) 了解C语言编译器的命名方法，主、子程序之间参数传递的机制。

# 2 实验内容

任务1. 宏与子程序设计

进一步修改与增强实验一任务4的**网店商品信息管理程序**的功能，主要调整功能三。

1. 调整后的功能三的描述

(1) 首先显示一个功能菜单（格式自行定义。若是未登录状态，只显示菜单“1”和“6”）:

1=查询商品信息，2=修改商品信息，3=计算推荐度，

4=计算推荐度排名，5=输出全部商品信息，6=程序退出。

输入1-6的数字进入对应的功能。

(2) 查询商品信息

提示用户输入要查询的商品名称。若未能在网店中找到该商品，重新提示输入商品名称。若只输入回车，则回到功能三(1)。（思考一下模糊查询如何实现）

找到该商品之后，按照：“商品名称，折扣，销售价，进货总数，已售数量，推荐度”顺序显示该商品的信息。显示之后回到功能三(1)。

(3) 修改商品信息

提示用户输入要修改信息的商品名称。[若把接下来的处理步骤写成子程序，则商品名称（或其偏移地址）就是子程序的入口参数，是否找到、是否是回车或者修改成功的信息是出口参数]。若未能在网店中找到该商品，重新提示输入商品名称。若只输入回车，则回到功能三(1)。

找到该商品之后，按照：折扣，进货价，销售价，进货总数的次序，逐一先显示原来的数值，然后输入新的数值（若输入有错，则重新对该项信息进行显示与修改。若直接回车，则不修改该项信息）。

如：折扣：9》8 //符号“》”仅作为分隔符，也可以选择其他分隔符号

进货价：25》24

销售价：46》5A6 //输入了非法数值，下一行重新显示和输入

销售价：46》56

进货总数：30》 //直接回车时，对这项信息不做修改

当对这些信息都处理完毕后，回到功能三(1)。

(4) 计算推荐度

从头到尾依次将每个商品的推荐度计算出来。回到功能三(1)。

(5) 计算推荐度排名

对SHOP中的每个商品按照推荐度的大小排名，排名信息可以存放到自行定义的一组结构变量中。回到功能三(1)。

(6) 输出全部商品信息

将SHOP中的所有商品信息显示到屏幕上，包括排名。具体的显示格式自行定义（可以按照存放次序显示，也可以按照商品推荐度排名的次序显示，等等，显示方式可以作为子程序的入口参数）。回到功能三(1)。

2.其他要求

(1) **两人一组**，一人负责包括菜单显示、程序退出在内的主程序，以及菜单中的功能(1)和(2)；另一人负责菜单中的功能(3)、(4)和(5)。各自汇编自己的模块，设计测试方法，测试通过；然后把自己的模块交给对方，各自把对方的程序整合到自己的程序中，连接生成一个程序，再进行整体调试。

实验报告中只需要描述自己负责的相关功能的设计思想、流程图、源程序。但在设计思想中要描述整体框架（包括整体的模块结构图、功能模块与子程序之间的对应关系等）和分工说明（包括模块的分配，两人协商一致的函数名、变量名等信息）。实验步骤和记录中要描述自己功能的实现与测试以及与同组模块整合后的联调与测试。

**注意，**在每个模块的开始，注明编写者的名字以及同组同学的名字。整合到一起时，要注意删掉自己测试时额外增加的代码，若有重复的模块（如：因两个人都会使用进制转换程序，导致各自模块中可能都有相同的进制转换子程序），也需要去掉重复的部分。

**建议分组方法：**按照学号（或前后左右相邻座位号）顺序依次两人一组，若班级人数为奇数，则最后三人一组（其中两人的分工是相同的，第三人只需要选择其中一个同学的模块与自己模块整合即可）。

(2) 排名的基本要求是按照推荐度从高到低计算名次，也可以考虑按照指定字段（比如已售数量等）排名。相同推荐度排名相同，下一个相邻推荐度的名次应该是排名在前的所有商品种类“和”的下一个数值。

(3) 将9号和10号DOS系统功能调用定义成宏指令并调用。功能(1)–(5)应尽量采用子程序方式实现。需要借鉴书上的进制转换程序：十进制转二进制的子程序F10T2和二进制转十进制的子程序F2T10（可以复用网站上相关的代码）。

任务2. 在C语言程序中调用汇编语言实现的函数

对于任务1的程序进行改造，主控程序、以及输入输出较多的某一个功能（如功能（1）、（2）、（5）中的某一个）用C语言实现，其他功能用独立的汇编语言子程序的方式实现； 在C语言程序中调用汇编语言子程序。

提示：本任务不分组，但要利用任务1自己整合后的结果。

# 3 实验过程

## 3.1 任务1

### 3.1.1 设计思路

本次实验中，我主要完成了主程序和功能(1)和(2)。主程序中主要实现了对各函数的调度功能，其中，我完成了用于显示菜单的函数ShowMenu以及退出程序的函数ExitProgram，另外，我还设计了函数InputString和ShowString将9号和10号中断的功能进行封装，用于对字符串的读取和输出。

主函数中首先调用相应的函数完成欢迎信息的输出以及用户登录的功能，并确定用户类型，然后根据用户类型输出相应的目录，并提示用户选择相应的功能，并将程序跳转到执行相应功能的代码部分，若用户输入非法的字符，则程序将提示用户输入错误并要求其重新输入。

对于功能1，首先提示用户输入待查找的商品名称，然后将该商品名与已有的商品名逐个比较，直到找到该商品。若找到该商品，则调用outone函数(该函数由西月栋设计，用于输出一个商品的全部信息)输出该商品的全部信息，若未找到，则输出错误提示信息，并要求用户重新输入。

对于功能2，首先与功能1相同，找到待修改的商品名称。若找到该商品，则调用change\_good函数(该函数由西月栋设计，用于修改商品信息)，若未找到，则输出错误提示信息，并要求用户重新输入。

### 3.1.2 流程图

程序流程图如图3-1所示。



图3-1 程序流程图

### 3.1.3 源代码

本次实验中，我完成的代码在main.asm文件中，其中的源代码如下：

extrn outone:far, sort:far, outall:far, change\_good:far

public io\_1, io\_2, io\_3, io\_4, io\_5, io\_6, io\_7, io\_8, BUF1, bufF2T10, DAT, SIGN

.386

goodStruct struct

good\_name DB 10 dup(0)

good\_len DB 0

good\_sale DB 10

good\_in\_price DW ?

good\_out\_price DW ?

good\_all\_count DW ?

good\_sale\_count DW ?

good\_recommamd DW ?

good\_sort DW 1

goodStruct ends

data segment use16

bname DB 'huao', 6 dup(0), 4 ;boss name

bpass DB 'huao', 6 dup(0), 4 ;password

N EQU 10 ;number

shopname DB 'huao$' ;shop name

auth DB ?

numOfBag EQU 1000

ga1 goodStruct <'pen$', 3, 10, 35, 56, 70, 25, ?>

ga2 goodStruct <'book$', 4, 9, 12, 30, 25, 5, ?>

goodStruct N-3 dup(<'egg$', 3, 7, 35, 56, 70, 25, ?>)

bag goodStruct <'bag$', 3, 10, 20, 50, numOfBag, 0, ?>

goodLen EQU $ - bag

welcomeMsg DB 'welcome to ', '$'

welcomeShopMsg DB ' shop$'

loginErrorMsg DB 'Login Error', 0ah, 0dh, '$'

outenter DB 0ah, 0dh, '$'

info DB 'input your name and password', 0ah, 0dh, '$'

in\_name DB 10, ?, 10 dup(0), '$'

in\_pass DB 6, ?, 6 dup(0), '$'

checkname DB 'input commodity you want to check:', '$'

goodname DB 10, ?, 10 dup(0)

notFound DB 'good not found', 0ah, 0dh, '$'

menu1 DB '1. Query product information', 0ah, 0dh, '$'

menu2 DB '2. Edit product information', 0ah, 0dh, '$'

menu3 DB '3. Calculate recommendation', 0ah, 0dh, '$'

menu4 DB '4. Calculate recommendation ranking', 0ah, 0dh, '$'

menu5 DB '5. Output all product information', 0ah, 0dh, '$'

menu6 DB '6. exit the program', 0ah, 0dh, '$'

errorInputMsg DB 'Input Error. Input again.', 0ah, 0dh, '$'

io\_1 db 'good\_name is: $'

io\_2 db 'good\_sale is: $'

io\_3 db 'good\_in\_price is: $'

io\_4 db 'good\_out\_price is: $'

io\_5 db 'good\_all\_count is: $'

io\_6 db 'good\_sale\_count is: $'

io\_7 db 'good\_recommand is: $'

io\_8 db 'good\_sort is: $'

BUF1 db 10, ?, 10 dup(0)

bufF2T10 db 12 dup(0)

DAT DW 0AH

SIGN DB ?

data ends

stack segment use16 stack

DB 200 dup(0)

stack ends

code segment use16

WelcomeFunc PROTO near stdcall

InputString PROTO near stdcall stringInputAddr:WORD

ShowMenu PROTO near stdcall userType:BYTE

ShowString PROTO near stdcall stringAddr:WORD

ExitProgram PROTO near stdcall

quaryGood proto near stdcall

start:

mov ax, data

mov ds, ax

invoke WelcomeFunc

mov bx, offset auth

chooseFunc:

invoke ShowMenu, ds:[bx]

mov ah, 1

int 21h

mov bx, offset outenter

invoke ShowString, bx

;此时al中为功能数

cmp al, '1'

jz func1

cmp al, '2'

jz func2

cmp al, '3'

jz func3

cmp al, '4'

jz func4

cmp al, '5'

jz func5

cmp al, '6'

jz func6

mov bx, offset errorInputMsg

invoke ShowString, bx

jmp chooseFunc

func1:

invoke quaryGood

jmp start

func2:

push di

mov bx, offset checkname

invoke ShowString, bx

mov bx, offset goodname

invoke InputString, bx

mov bx, offset outenter

invoke ShowString, bx

mov si, offset goodname

mov cl, [si + 1]

cmp cl, 0

jz start

call findgood

cmp ch, -1

jz printNotFound2

mov di, bx

call change\_good

pop di

jmp start

printNotFound2:

mov dx, offset notFound

invoke ShowString, dx

jmp func2

func3:

call calIndex

jmp start

func4:

mov ax, N

mov si, offset ga1

call sort

jmp start

func5:

mov ax, N

mov si, offset ga1

call outall

jmp start

func6:

invoke ExitProgram

quaryGood proc near stdcall

quaryGoodBegin:

mov bx, offset checkname

invoke ShowString, bx

mov bx, offset goodname

invoke InputString, bx

mov bx, offset outenter

invoke ShowString, bx

mov si, offset goodname

mov cl, [si + 1]

cmp cl, 0

jz EndJmpStart

call findgood

cmp ch, -1

jz printNotFound

mov si, offset auth

mov cl, [si]

cmp cl, 1

jz authOK

jmp authNotOK

authOK:

push si

mov si, bx

call outone

pop si

jmp EndJmpStart

authNotOK:

push si

mov si, bx

call outone

pop si

jmp EndJmpStart

printNotFound:

mov dx, offset notFound

invoke ShowString, dx

jmp quaryGoodBegin

EndJmpStart:

ret

quaryGood endp

WelcomeFunc PROC near stdcall uses ax bx dx

welcomeStart:

mov ax, offset welcomeMsg

invoke ShowString, ax

mov ax, offset shopname

invoke ShowString, ax

mov ax, offset welcomeShopMsg

invoke ShowString, ax

mov ax, offset outenter

invoke ShowString, ax

mov ax, offset info

invoke ShowString, ax

mov ax, offset in\_name

invoke InputString, ax

mov ax, offset outenter

invoke ShowString, ax

;下面将读入的回车符号置为0

mov si, offset in\_name

mov al, ds:[si + 1]

mov ah, 0

mov bx, ax

mov ax, 0

mov ds:[si + bx + 2], al

mov al, ds:[si + 1]

cmp al, 0

jz welcomeUser

cmp al, 1

jz checkQuit

jmp next3

checkQuit:

mov al, ds:[si + 2]

cmp al, 71h

invoke ExitProgram

next3:

mov dx, offset in\_pass

invoke InputString, dx

mov si, offset in\_pass

mov al, ds:[si + 1]

mov ah, 0

mov bx, ax

mov ax, 0

mov ds:[si + bx + 2], al

mov dx, offset outenter

invoke ShowString, dx

lea bx, in\_name[2]

mov si, offset in\_name

mov al, ds:[si + 1]

mov si, offset bname

cmp al, ds:[si + 10]

jnz errorLogin

call compareName

cmp ch, 1

jnz errorLogin

lea bx, in\_pass[2]

mov si, offset in\_pass

mov al, ds:[si + 1]

mov si, offset bpass

cmp al, ds:[si + 10]

jnz errorLogin

call comparePass

cmp ch, 1

jnz errorLogin

mov al, 1

mov si, offset auth

mov [si], al

ret

errorLogin:

lea dx, loginErrorMsg[0]

mov ah, 9

int 21h

jmp welcomeStart

welcomeUser:

mov al, 0

mov si, offset auth

mov [si], al

ret

WelcomeFunc ENDP

InputString PROC near stdcall uses dx ax, stringInputAddr:WORD

mov dx, stringInputAddr

mov ah, 10

int 21h

ret

InputString ENDP

ShowMenu PROC near stdcall uses ax, userType:BYTE

mov ax, offset menu1

invoke ShowString, ax

cmp userType, 1

jnz user

mov ax, offset menu2

invoke ShowString, ax

mov ax, offset menu3

invoke ShowString, ax

mov ax, offset menu4

invoke ShowString, ax

mov ax, offset menu5

invoke ShowString, ax

user:

mov ax, offset menu6

invoke ShowString, ax

ret

ShowMenu ENDP

ShowString PROC near stdcall uses dx ax, stringAddr:WORD

mov dx, stringAddr

mov ah, 9

int 21h

ret

ShowString ENDP

ExitProgram PROC near stdcall

mov ax, 4C00h

int 21h

ExitProgram ENDP

compareName proc

push si

compName:

cmp al, 0

jz trueName

dec al

mov ah, 0

mov di, ax

mov si, offset bname

add si, di

mov dl, [si]

sub dl, [bx + di]

jz compName

mov ch, 0

pop si

ret

trueName:

mov ch, 1

pop si

ret

compareName endp

comparePass proc

push si

compPass:

cmp al, 0

jz truePass

dec al

mov ah, 0

mov di, ax

mov si, offset bpass

add si, di

mov dl, [si]

sub dl, [bx + di]

jz compPass

mov ch, 0

pop si

ret

truePass:

mov ch, 1

pop si

ret

comparePass endp

findgood proc

push di

mov di, offset goodname

mov bl, [di + 1]

mov bh, 0

mov si, bx

mov di, offset ga1

lea bx, [di]

sub bx, goodLen

mov dx, N

inc dx

goodLoop:

dec dx

add bx, goodLen

cmp dx, 0

jz next2

mov al, [bx + 10]

mov ah, 0

cmp ax, si

jnz goodLoop

call compare

cmp ch, 1

jz foundgood

jmp goodLoop

next2:

mov ch, -1

pop di

ret

foundgood:

mov ch, bh

pop di

ret

findgood endp

compare proc

push si

comp:

cmp al, 0

jz found

dec al

mov ah, 0

mov di, ax

mov si, offset goodname

add si, di

mov dl, [si + 2]

sub dl, [bx + di]

jz comp

mov ch, 0

pop si

ret

found:

mov ch, 1

pop si

ret

compare endp

calIndex proc

pusha

mov si, offset ga1

lea bx, [si]

sub bx, goodLen

mov di, N

inc di

calLoop:

dec di

add bx, goodLen

cmp di, 0

jz calEnd

mov ax, [bx].good\_in\_price

mov cx, 1280

mul cx

push dx

push ax

mov al, byte ptr [bx].good\_out\_price

mov cl, [bx].good\_sale

mul cl

mov cx, ax

pop ax

pop dx

div cx

push ax

mov ax, [bx].good\_sale\_count

mov cx, 64

mul cx

mov cx, [bx].good\_all\_count

div cx

mov dx, ax

pop ax

add ax, dx

cmp ax, 100

jns recommA

cmp ax, 50

jns recommB

cmp ax, 10

jns recommC

jmp recommF

recommA:

push ax

mov ax, 'A'

mov [bx + 20], al

pop ax

jmp calLoop

recommB:

push ax

mov ax, 'B'

mov [bx + 20], al

pop ax

jmp calLoop

recommC:

push ax

mov ax, 'C'

mov [bx + 20], al

pop ax

jmp calLoop

recommF:

push ax

mov ax, 'F'

mov [bx + 20], al

pop ax

jmp calLoop

calEnd:

popa

ret

calIndex endp

code ends

end start

### 3.1.4 实验步骤

使用visual studio code编写代码，并在dosbox中完成相应的编译、链接和调试过程。本实验中包含3个源文件，分别为main.asm, sort.asm以及F10T2W.asm。将三个文件使用masm分别编译，得到三个.obj的文件，然后将这三个obj文件使用工具link链接起来，得到可执行文件，使用td工具打开该文件即可对其进行调试过程。

### 3.1.5 实验记录和分析

1. 对菜单功能的测试

当用户输入正确的用户名和密码时，程序将给出功能菜单，显示6个功能，当用户为输入用户名和密码时，程序将给出2个功能的功能菜单，其测试如图3-2和图3-3所示。

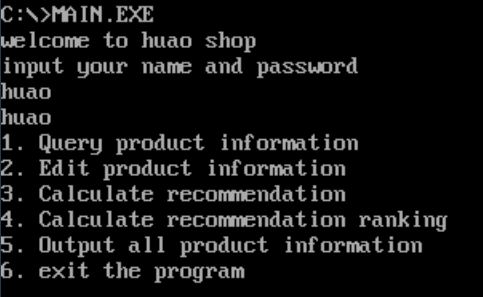


图3-2 老板查看的功能菜单

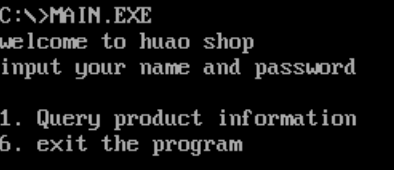


图3-3 用户查看的功能菜单

2. 查询商品信息功能测试

当用户选择功能1时，程序执行查询商品信息的功能，当用户输入一个存在的商品名称时，程序将输出该商品的全部信息，若该商品不存在，则程序提醒用户该商品不存在，并要求其重新输入。其测试截图如图3-4和图3-5所示。

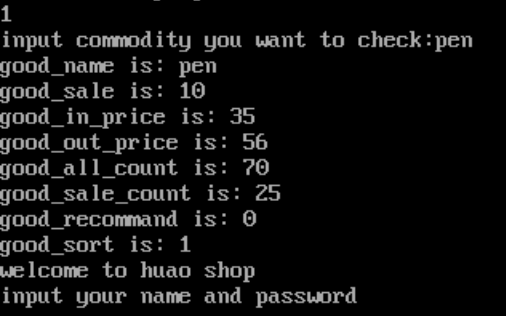


图3-4 当商品存在时，查询商品信息

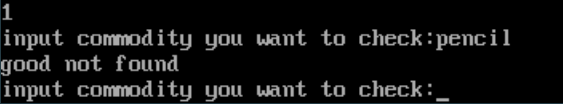


图3-5 当商品不存在时，查询商品信息

3. 修改商品信息功能测试

当用户选择功能2时，程序将执行修改商品信息的功能，当该商品存在时，程序将逐个的输出该商品的信息并要求用户输入该信息修改后的值，若该商品不存在，则程序将输出该商品不存在并提示用户重新输入商品名称，其测试截图如图3-6和3-7所示。

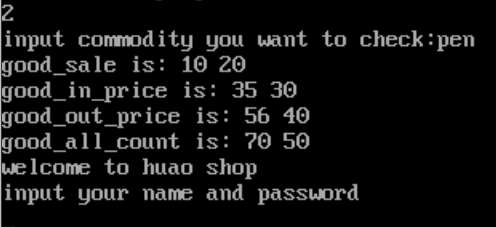


图3-6 商品存在时，修改商品信息

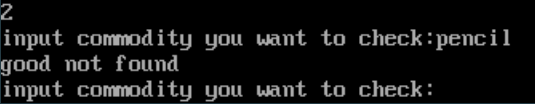


图3-7 商品不存在时，修改商品信息

4. 退出程序功能测试

当用户选择功能6时，程序将退出，其测试截图如图3-8所示。

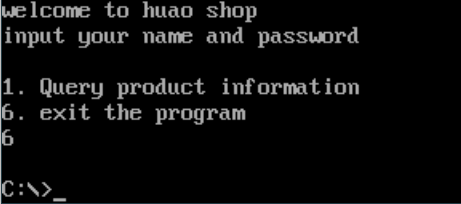


图3-8 退出程序功能测试截图

## 3.2 任务2

### 3.2.1 设计思路

此任务中，使用C语言改写任务一中的汇编主程序，并将其中查找商品的功能使用C语言函数实现，另外，由于本次任务在windows环境下实现，因此使用windows api函数以及C语言库函数替代原本在任务一中使用的dos中断功能。

### 3.2.2 流程图

此任务中程序的流程与任务一中的程序相同，仅仅对其中的实现方法进行了相应的修改，因此程序流程图与任务一相同，见图3-1。

### 3.2.3 源代码

此任务中，对原任务一中的汇编函数修改较少，其中修改部分如下。

1. 添加对待使用C语言函数及windows api的声明

printf proto cdecl :dword

scanf proto cdecl :dword, :dword

strlen proto cdecl :dword

ExitProcess proto stdcall :dword

putchar proto cdecl :byte

2. 对输入输出函数的修改

此任务中使用C语言printf、putchar和scanf函数实现输入输出功能，以替代dos下的9号和10号中断功能。因此对函数InputString、宏gets、宏puts、宏CRLF以及函数showString做如下修改。

ShowString PROC stdcall stringAddr:DWORD

invoke printf, stringAddr

ret

ShowString ENDP

InputString PROC stdcall uses edx eax, stringInputAddr:DWORD

mov eax, stringInputAddr

lea edx, scanfArgStr

add eax, 2

invoke scanf, edx, eax

mov dl, 10

mov eax, stringInputAddr

mov [eax], dl

add eax, 2

invoke strlen, eax

mov edx, stringInputAddr

mov [edx + 1], al

mov [edx + 12], 0

ret

InputString ENDP

puts macro ofset:req

push eax

mov eax, offset ofset

invoke printf, eax

pop eax

endm

;input change cx si,cx is length, si is addr

gets macro ofset:req

lea esi, ofset

invoke InputString, esi

mov cl, [esi + 1]

movsx cx, cl

add esi, 2

endm

CRLF macro

invoke putchar, 0ah

invoke putchar, 0dh

endm

3. 对退出程序函数的修改

程序退出函数使用windows api完成而不是dos中断，更改后的函数如下

ExitProgram PROC

invoke ExitProcess, 0

ret

ExitProgram ENDP

本次任务中添加了C语言程序，其源代码如下

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define N 10 //商品数量

#define numOfBag 1000

typedef struct goodStruct {

char good\_name[10];

char good\_len;

char good\_sale;

short good\_in\_price;

short good\_out\_price;

short good\_all\_count;

short good\_sale\_count;

short good\_recommand;

short good\_sort;

} goodStruct;

char bname[11] = "huao";

char bpass[11] = "huao";

char shopname[] = "huao";

char auth;

goodStruct good[N];

int goodLen = sizeof(goodStruct);

char welcomeMsg[] = "welcome to ";

char welcomeShopMag[] = " shop";

char loginErrorMsg[] = "Login Error\n";

char outenter[] = "\n";

char info[] = "input your name and password\n";

char in\_name[13];

char in\_pass[9];

char checkname[] = "input commodity you want to check:";

char goodname[12];

char notFound[] = "good not found\n";

char menu1[] = "1. Query product information\n";

char menu2[] = "2. Edit product information\n";

char menu3[] = "3. Calculate recommendation\n";

char menu4[] = "4. Calculate recommendation ranking\n";

char menu5[] = "5. Output all product information\n";

char menu6[] = "6. exit the program\n";

char errorInputMsg[] = "Input Error. Input again.\n";

char bufF2T10[12];

short DAT = 10;

char SIGN;

extern "C" void \_cdecl WelcomeFunc(char \*welcomeMsg, char \* shopname, char \*welcomeShopMsg, char \*outenter, char \* info, char \*in\_name, char \*in\_pass, char \*bname, char \*bpass, char \*auth, char \*loginErrorMsg);

extern "C" void \_cdecl ShowMenu(char auth, char \*menu1, char \*menu2, char \*menu3, char \*menu4, char \*menu5, char \*menu6);;

extern "C" void \_cdecl calIndex(void \*goodAddr, int goodLen, short goodNum);;

extern "C" void \_cdecl change\_good(char \*goodAddr, char \*SIGN, short \*DAT);

extern "C" void \_cdecl outone(char \*goodAddr);

extern "C" void \_cdecl outall(short goodSum, void \*goodAddr);

extern "C" void \_cdecl sort(short goodSum, void \*goodAddr);

extern "C" void \_cdecl ExitProgram(void);

void init(void);

char \*findGood(char \*goodName);

int main(void)

{

init();

char temp;

int funcNum;

char \*goodAddr;

while (1)

{

WelcomeFunc(welcomeMsg, shopname, welcomeShopMag, outenter, info, in\_name, in\_pass, bname, bpass, &auth, loginErrorMsg);

ShowMenu(auth, menu1, menu2, menu3, menu4, menu5, menu6);

printf("Please choose a function:");

scanf("%d", &funcNum);

switch (funcNum)

{

case 1:

while (1)

{

printf(checkname);

scanf("%s", goodname);

goodAddr = findGood(goodname);

if (goodAddr == NULL)

{

printf(notFound);

}

else

{

break;

}

}

outone(goodAddr);

break;

case 2:

while (1)

{

printf(checkname);

scanf("%s", goodname);

goodAddr = findGood(goodname);

if (goodAddr == NULL)

printf(notFound);

else

break;

}

change\_good(goodAddr, &SIGN, &DAT);

break;

case 3:

calIndex(good, goodLen, N);

break;

case 4:

sort(N, good);

break;

case 5:

outall(N, good);

break;

case 6:

ExitProgram();

break;

default:

printf("input error, please input again\n");

break;

}

}

printf("\n");

scanf("%c", &temp);

getchar();

strlen(bname);

return 0;

}

char \*findGood(char \*goodName)

{

void \*goodAddr = good;

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

{

if (strcmp(goodName, good[i].good\_name) == 0)

{

return (char \*)goodAddr + i \* goodLen;

}

}

return NULL;

}

void init(void) {

bname[10] = 4;

bpass[10] = 4;

strcpy(good[0].good\_name, "pen");

good[0].good\_len = 3;

good[0].good\_sale = 10;

good[0].good\_in\_price = 35;

good[0].good\_out\_price = 56;

good[0].good\_all\_count = 70;

good[0].good\_sale\_count = 25;

good[0].good\_sort = 1;

strcpy(good[1].good\_name, "book");

good[1].good\_len = 4;

good[1].good\_sale = 9;

good[1].good\_in\_price = 12;

good[1].good\_out\_price = 30;

good[1].good\_all\_count = 25;

good[1].good\_sale\_count = 5;

good[1].good\_sort = 1;

for (int i = 2; i < N - 1; i++)

{

strcpy(good[i].good\_name, "egg");

good[i].good\_len = 3;

good[i].good\_sale = 7;

good[i].good\_in\_price = 35;

good[i].good\_out\_price = 56;

good[i].good\_all\_count = 70;

good[i].good\_sale\_count = 25;

good[i].good\_sort = 1;

}

strcpy(good[N - 1].good\_name, "bag");

good[N - 1].good\_len = 3;

good[N - 1].good\_sale = 10;

good[N - 1].good\_in\_price = 20;

good[N - 1].good\_out\_price = 50;

good[N - 1].good\_all\_count = numOfBag;

good[N - 1].good\_sale\_count = 0;

good[N - 1].good\_sort = 1;

}

### 3.2.4 实验步骤

本次实验在windows环境下完成，使用visual studio完成了代码的编写、C语言与汇编语言的链接以及代码的运行和调试过程。本次任务中一共包含四个文件，分别为main.cpp, func1.asm, sort.asm以及F10T2.asm，其中main.cpp为C语言源程序，func1.asm包含了任务一中main.asm中包含的函数，sort.asm和F10T2.asm与任务一中的文件类似，仅修改了部分与系统相关的内容，具体的修改部分见3.2.3节。

首先，在windows的控制台中使用ml /I . /Zm /c /Ta [文件名]对汇编源文件进行编译，得到.obj的文件，然后将得到的三个.obj文件添加到visual studio的项目中，使用visual studio生成解决方案，即可得到相应的可执行文件。

另外，在visual studio中使用调试工具可直接对该可执行文件进行汇编指令级的调试。

### 3.2.5 实验记录和分析

本次实验中主要实现了对六个功能的模块化设计，对六个功能的测试结果如下。

如图3-9和3-10为对查询商品功能的测试截图。其中图3-9为商品存在时的情况，图3-10为商品不存在的情况。

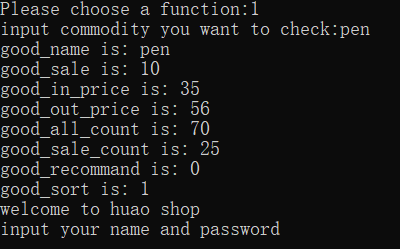


图3-9 商品存在时，查询商品功能测试

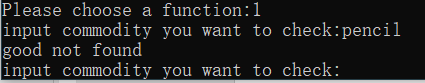


图3-10 商品不存在时，查询商品功能测试

如图3-11为修改商品信息功能的测试截图。

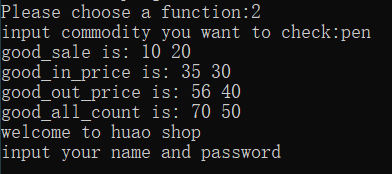


图3-11 修改商品信息功能测试

执行功能3、4后，程序没有明确输出，但是已对内存中相应的数据进行了修改，当程序执行功能3和4后，再次调用功能1，可以得知，程序对商品的推荐度做了计算并根据推荐度给出了各商品的排名，再次调用功能1后的程序截图如图3-12所示。

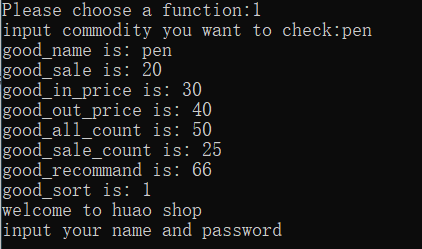


图3-12 调用功能3、4后调用功能1的输出截图

如图3-13为输出全部商品信息的测试截图。

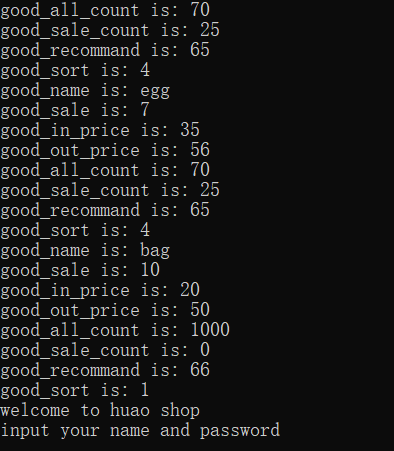


图3-13 输出全部商品信息功能测试

如图3-14为退出程序功能测试。选择功能6后，程序直接退出，并返回到控制台。



图3.14 退出程序功能测试

# 4 总结和体会

在本次实验的任务一中，我和小组成员西月栋各自完成了源代码的编写，在编写源代码以及各自编译的过程中，由于前几次的实验基础，我们可以很好的完成，但是在将文件链接的时候，我们遇到了很多问题。例如在多个文件中均定义了数据段，在链接后访问数据段元素时导致访问位置与预期不相符以及调用另一个文件中的函数时出现的问题，在参阅课本并且做出相应的尝试后，我们最终解决了这些问题，并完成了任务一的要求。

通过完成任务一，我对汇编语言的链接有了更多的认识，同时也对汇编语言编译过程中对变量、函数等的处理有了更深刻的了解。

在任务二中，我同样遇到了很多的链接时出现的错误，在经过对代码的仔细分析和修正后，最终将这些问题一一解决。在这一过程中，我直观的认识到了不同的函数调用方式对于出现执行方式的影响，同时也对C语言的编译过程以及编译结果有了新的认识，在对链接错误的修改过程中，我发现即使C语言中包含了stdio.h中所有函数的声明，但是只有其中被使用过的函数的定义会被编译并出现在.obj文件中，对于未被使用的函数，则编译器并不会处理它的定义部分。

通过本次实验，我对C语言和汇编语言的编译、链接过程有了更深入的认识，也给我在编写代码的过程中避免出错提供了大量的经验；另外，本次实验也让我对模块化程序设计有了更加正确的认识，让我了解到了很多在分模块编写程序时应该注意的问题。