**Department of Computer Science**

**CSCL2201 – Computer Organization and Assembly**

**Language**

**“Smart Cart System”**

**Project Report**

**Semester: Fall 2024**

**Group members**:

**1. Abdul Rehman 2312143**

**2. Muhammad Kumail Muhmmad Ameen 2312158**

**3. Syed Hisham Ali 2312179**

**4. Rawail Ahmed 2312170**

**Submitted to:**

**Sir Mubeen Ahmed Khan**

**Lab Engineer**

**Sir Waseem Rauf**

**TABLE OF CONTENT**

|  |  |  |
| --- | --- | --- |
| S.NO | CONTENT | PAGE NUMBERS |
| 1 | **OBJECTIVE** | **3** |
| 2 | **Description** | **4** |
| 3 | **Concepts Implemented** | **5** |
| 4 | **Project Features** | **6** |
| 5 | **SOURCE CODE** | **7-30** |
| 6 | **OUTPUT** | **31-32** |
| 7 | **CONCLUSION** | **33** |

**Objective**

This project's main objective is to use x86 Assembly Language to create and implement an interactive smart cart system that mimics a simple shopping cart user friendly experience. The application dynamically computes the entire cost while letting users choose the category of their products then choosing the product, entering the quantities of their selected product. With features like dynamic bill calculation, error management for invalid inputs, and a comprehensive final bill summary of the user's purchases, this project mimics the operation of a retail checkout procedure. This project shows how assembly language may be used to create a useful, user-driven application by utilising low-level programming techniques.

**Description**

Users can browse through predefined product categories that are Clothing, Shoes, Dairy, and Meat, each of which contains many goods, in this text-based simulation of a shopping management system. Users choose products and enter the quantities to add to their carts using the system's user-friendly menu-driven navigation. The application stores information about every purchase, prints the present amount that is the multiplication of the current product and quantity and then dynamically updates the overall cost based on the chosen goods and quantities up till now. It creates a thorough bill summary at the conclusion of the session, that only shows the product that is bought and the quantity that was chosen previously. Additionally, the application has screen management to keep the interface neat and orderly and error handling to guarantee the correctness of input.

**Concepts Implemented**

The project makes use of a number of fundamental assembly language programming ideas, such as procedural programming, dynamic pricing and computation, and user input management. In order to verify user input and show error messages for invalid options and ask again for input, it integrates error handling. The application ensures an engaging and aesthetically pleasing user experience by manipulating the screen, displaying text, clearing the screen and placing the cursor

Back via BIOS and DOS interrupts. Counters and pre-specified memory addresses for every item are used to handle data storage, enabling real-time tracking of purchased quantities. Additionally, by using reusable procedures for tasks like clear screen, error messaging, and amount input, the project showcases the usage of macros and modular design.

**Project Features**

* Users can explore specified categories and commodities using the interactive menu provided by this smart cart system.
* It maintains counters for every product to keep track of purchases and allows dynamic total by determining the cost of things based on user-specified amounts.
* The application offers thorough error handling, guarantees that users don't enter invalid data, and shows error notifications when necessary.
* The technology creates a thorough bill summary with itemised quantities and total expenses at the conclusion of the session.
* In order to improve the user experience, it also integrates screen management tools to remove extraneous information and offer a tidy interface (CLEAR SCREEN FUNCTION).
* A final bill display and continuous shopping capabilities allow the system to simulate a full retail checkout process.

**Source Code**

convert1 macro present,tot

lea dx,present

mov ah,9

int 21h

mov ax,tot

call cc

endm

.model small

.stack 100h

.data

heading db 10,13, ' WELCOME TO SMART CART SYSTEM ',10,13 , ' $'

nameandroll db ' Made by :',10,13,' Muhammad Kumail 2312158 ' ,10,13 , ' Abdul Rehman 2312143 ' , 10,13,' Syed Hisham Hashmi 2312179 ' , 10,13 , ' Rawail Ahmed 2312170 ' , 10,13,10,13,'$'

category db 10,13, ' SELECT THE FOLLOWING CATEGORY IN WHICH THE PRODUCT LIES IN : $'

item db 10,13, ' SELECT THE PRODUCT YOU WANT TO BUY : $'

clothes db 10,13, ' 1 : CLOTHES' , 10,13,'$'

Shoes db 10,13, ' 2 : SHOES' , 10,13,'$'

Dairy db 10,13, ' 3 : DAIRY' , 10,13,'$'

meat db 10,13, ' 4 : MEAT' , 10,13,'$'

Intro db 10,13 ,' ITEM PRICE ', 10,13,'$'

c1 db 10,13,' 1 : CASUAL SHIRT(MALE) USD 10 ',10,13, ' $'

c2 db 10,13,' 2 : CASUAL SHIRT(FEMALE) USD 20 ',10,13, ' $'

c3 db 10,13,' 3 : PANT(MALE) USD 50 ',10,13, ' $'

c4 db 10,13,' 4 : PANT(FEMALE) USD 70 ',10,13, ' $'

s1 db 10,13,' 1 : CASUAL SHOES(MALE) USD 80 ',10,13, ' $'

s2 db 10,13,' 2 : CASUAL SHOES(FEMALE) USD 120 ',10,13, ' $'

s3 db 10,13,' 3 : PREDATOR USD 150 ',10,13, ' $'

s4 db 10,13,' 4 : JORDANS USD 170 ',10,13, ' $'

d1 db 10,13,' 1 : MILK USD 4 ',10,13, ' $'

d2 db 10,13,' 2 : CHEESE USD 1 ',10,13, ' $'

d3 db 10,13,' 3 : EGG USD 2 ',10,13, ' $'

d4 db 10,13,' 4 : BREAD USD 1 ',10,13, ' $'

m1 db 10,13,' 1 : CHICKEN USD 4 ',10,13, ' $'

m2 db 10,13,' 2 : LAMB USD 5 ',10,13, ' $'

m3 db 10,13,' 3 : BEEF USD 6 ',10,13, ' $'

m4 db 10,13,' 4 : BUFFALO USD 7 ',10,13, ' $'

error db 10,13,'WRONG INPUT, TRY BETWEEN 1-4 , TRY AGAIN ' ,10,13, "$"

qua db 10,13,' ENTER THE QUANTITY OF THE PRODUCT $'

present1 db 10,13,'PRESENT AMOUNT IS : USD $ '

again1 db 10,13,' DO YOU WANT TO BUY MORE <1> YES <2>NO $'

totalamount dw 0

countcsm dw 0 ;1

countcsf dw 0 ;2

countpant dw 0 ;3

countfpant dw 0 ;4

countshoesm dw 0 ;1

countshoesf dw 0 ;2

countpred dw 0 ;3

countjord dw 0 ;4

countmilk dw 0 ;1

countcheese dw 0 ;2

countegg dw 0 ;3

countbread dw 0 ;4

countchick dw 0 ;1

countlamb dw 0 ;2

countbeef dw 0 ;3

countbuff dw 0 ;

countcsmdis db 10,13,'TOTAL CASUAL SHIRT(MALE) BOUGHT ARE : $'

countcsfdis db 10,13,'TOTAL CASUAL SHIRT(FEMALE) BOUGHT ARE : $'

countpantmdis db 10,13,'TOTAL PANT(MALE) BOUGHT ARE : $'

countpantfdis db 10,13,'TOTAL PANT(FEMALE) BOUGHT ARE : $'

countshoesmdis db 10,13,'TOTAL CASUAL SHOES(MALE) BOUGHT ARE : $'

countshoesfdis db 10,13,'TOTAL CASUAL SHOES(FEMALE) BOUGHT ARE : $'

countpreddis db 10,13,'TOTAL PREDATORS BOUGHT ARE : $'

countjorddis db 10,13,'TOTAL JORDANS BOUGHT ARE : $'

countmilkdis db 10,13,'TOTAL LITRES OF MILK BOUGHT ARE : $'

countcheesedis db 10,13,'TOTAL PACKETS OF CHEESE BOUGHT ARE : $'

counteggdis db 10,13,'TOTAL DOZEN OF EGG BOUGHT ARE : $'

countbreaddis db 10,13,'TOTAL PACKETS OF BREAD BOUGHT ARE : $'

countchickdis db 10,13,'TOTAL Kg''s CHICKEN BOUGHT ARE : $'

countlambdis db 10,13,'TOTAL Kg''s LAMB BOUGHT ARE $'

countbeefdis db 10,13,'TOTAL Kg''s BEEF BOUGHT ARE $'

countbuffdis db 10,13,'TOTAL Kg''s BUFFALO BOUGHT ARE $'

num dw 0

a dw 0

b dw 0

q dw 0

summ db 10,13,'THE BILL SUMMARY IS ',10,13,10,13,'$'

total db 10,13,10,13, 'THE TOTAL BILL IS : USD $'

.code

main proc

mov ax,@data

mov ds,ax

mov ah,9

mov bl,0F0h ;COLOR CODE

int 10H

lea dx , nameandroll

mov ah,9

int 21h

lea dx,heading

mov ah,9

int 21h

;print all the categories

CatLabel:

lea dx,clothes

mov ah,9

int 21h

lea dx,Shoes

mov ah,9

int 21h

lea dx,Dairy

mov ah,9

int 21h

lea dx,meat

mov ah,9

int 21h

lea dx,category

mov ah,9

int 21h

mov ah,1

int 21h

;if press 1

cmp al,'1'

je cloth1

;if press 2

cmp al,'2'

je shoes1

;if press 3

cmp al,'3'

je dairy1

;if press 4

cmp al,'4'

je meat1

call ClearScreenAndMoveCursor

lea dx ,error

mov ah,9

int 21h

jmp CatLabel

;label list all the available clothes

cloth1:

;print the intro

lea dx,Intro

mov ah,9

int 21h

;list all the available clothes

lea dx,c1

mov ah,9

int 21h

lea dx,c2

mov ah,9

int 21h

lea dx,c3

mov ah,9

int 21h

lea dx,c4

mov ah,9

int 21h

;select the item

call ask

cmp al,49

je csm

cmp al,50

je csf

cmp al,51

je pantm

cmp al,52

je pantf

call ClearScreenAndMoveCursor

call error1

jmp cloth1 ;if wrong return back to cloth menu

;label to list all the available shoes

shoes1:

lea dx,Intro

mov ah,9

int 21h

lea dx,s1

mov ah,9

int 21h

lea dx,s2

mov ah,9

int 21h

lea dx,s3

mov ah,9

int 21h

lea dx,s4

mov ah,9

int 21h

;ask to select item

call ask

cmp al,49

je cshoesm

cmp al,50

je cshoesf

cmp al,51

je pred

cmp al,52

je jordan

call ClearScreenAndMoveCursor

call error1

jmp shoes1 ;if wrong return back to shoes menu

;list all the available meat

meat1:

lea dx,Intro

mov ah,9

int 21h

lea dx,m1

mov ah,9

int 21h

lea dx,m2

mov ah,9

int 21h

lea dx,m3

mov ah,9

int 21h

lea dx,m4

mov ah,9

int 21h

call ask

cmp al,49

je chicken

cmp al,50

je lamb

cmp al,51

je beef

cmp al,52

je buff

call ClearScreenAndMoveCursor

call error1

jmp meat1

;list all the available dairy

dairy1:

lea dx,Intro

mov ah,9

int 21h

lea dx,d1

mov ah,9

int 21h

lea dx,d2

mov ah,9

int 21h

lea dx,d3

mov ah,9

int 21h

lea dx,d4

mov ah,9

int 21h

call ask

cmp al,49

je milk

cmp al,50

je cheese

cmp al,51

je egg

cmp al,52

je bread

call ClearScreenAndMoveCursor

call error1

jmp dairy1

;pricing

csm:

mov a,10

call quantity

mov cx,q

add countcsm,cx

jmp print

csf:

mov a,20

call quantity

mov cx,q

add countcsf,cx

jmp print

pantm:

mov a,50

call quantity

mov cx,q

add countpant,cx

jmp print

pantf:

mov a,70

call quantity

mov cx,q

add countfpant,cx

jmp print

cshoesm:

mov a,80

call quantity

mov cx,q

add countshoesm,cx

jmp print

cshoesf:

mov a,120

call quantity

mov cx,q

add countshoesf,cx

jmp print

pred:

mov a,150

call quantity

mov cx,q

add countpred,cx

jmp print

jordan:

mov a,170

call quantity

mov cx,q

add countjord,cx

jmp print

milk:

mov a,4

call quantity

mov cx,q

add countmilk,cx

jmp print

cheese:

mov a,1

call quantity

mov cx,q

add countcheese,cx

jmp print

egg:

mov a,2

call quantity

mov cx,q

add countegg,cx

jmp print

bread:

mov a,1

call quantity

mov cx,q

add countbread,cx

jmp print

chicken:

mov a,4

call quantity

mov cx,q

add countchick,cx

jmp print

lamb:

mov a,5

call quantity

mov cx,q

add countlamb,cx

jmp print

beef:

mov a,6

call quantity

mov cx,q

add countbeef,cx

jmp print

buff:

mov a,7

call quantity

mov cx,q

add countbuff,cx

jmp print

print:

mov ax,q

mul a

add totalamount,ax

mov a,ax

convert1 present1,a ;macro

jmp again

endprog:

mov ah,4ch

int 21h

again:

lea dx,again1

mov ah,9

int 21h

mov ah,1

int 21h

xor ah,ah ;single digit input thats why clear higher bits

mov b,ax ;save lower bits (higher bits is 0) before clearing screen

call ClearScreenAndMoveCursor

xor ah,ah

mov ax,b

cmp al,49

je CatLabel

cmp al,50

je t

call error1

jmp again

t:

lea dx,summ

mov ah,9

int 21h

;macro

l1:

cmp countcsm,0

je l2

convert1 countcsmdis,countcsm

l2:

cmp countcsf,0

je l3

convert1 countcsfdis,countcsf

l3:

cmp countpant,0

je l4

convert1 countpantmdis,countpant

l4:

cmp countfpant,0

je l5

convert1 countpantfdis,countfpant

l5:

cmp countshoesm,0

je l6

convert1 countshoesmdis,countshoesm

l6:

cmp countshoesf,0

je l7

convert1 countshoesfdis,countshoesf

l7:

cmp countpred,0

je l8

convert1 countpreddis,countpred

l8:

cmp countjord,0

je l9

convert1 countjorddis,countjord

l9:

cmp countmilk,0

je l10

convert1 countmilkdis,countmilk

l10:

cmp countcheese,0

je l11

convert1 countcheesedis,countcheese

l11:

cmp countegg,0

je l12

convert1 counteggdis,countegg

l12:

cmp countbread,0

je l13

convert1 countbreaddis,countbread

l13:

cmp countchick,0

je l14

convert1 countchickdis,countchick

l14:

cmp countlamb,0

je l15

convert1 countlambdis,countlamb

l15:

cmp countbeef,0

je l16

convert1 countbeefdis,countbeef

l16:

cmp countbuff,0

je t2

convert1 countbuffdis,countbuff

t2:

convert1 total,totalamount

jmp endprog

endp main

quantity proc

lea dx,qua

mov ah,9

int 21h

mov bl,10 ;store 10 in bl to mulitply

mov q,0 ;store present amount so initially store with 0

repeat:

mov ah,1

int 21h

cmp al,13

je done

sub al,48

xor ah,ah ;clear ax if any garbage value in ah

mov cx,ax ;temp store ax which we entered now in cx

mov ax,q ;total value pressed until before

mul bl ; ax = ax X bl(10) ;digits

add ax,cx ;add prev value to the multiplied one

mov q,ax

jmp repeat

ret

done:

ret

endp quantity

cc proc

xor dx,dx ;clears reg

mov bx,10 ;divisor

xor cx,cx

ll1:

div bx

push dx ;ax store qoutient dx store remainder

xor dx,dx

xor ah,ah

inc cx ;track cx

cmp ax,0 ;if no quotient left

jne ll1

ll2:

pop dx

add dx,48

mov ah,2

int 21h

loop ll2

ret

endp cc

ask proc

lea dx,item

mov ah,9

int 21h

mov ah,1

int 21h

ret

endp ask

error1 proc

lea dx,error

mov ah,9

int 21h

ret

endp error1

ClearScreenAndMoveCursor:

; Clear the screen (scroll up)

;80x25

mov al, 0 ; Number of lines to scroll , 0 to reset

mov bh, 0F0h ; Attribute (black background, white text)

mov ch, 0 ; Starting row (top)

mov cl, 0 ; Starting column (left)

mov dh, 24 ; Ending row (bottom)

mov dl, 79 ; Ending column (right)

mov ah, 6h ; Function to scroll up (clear screen)

int 10h ; Call BIOS video interrupt

; Move the cursor to the home position (top-left corner)

mov bh, 0 ; Page number (0 for default)

mov dh, 0 ; Row (0 for top)

mov dl, 0 ; Column (0 for left)

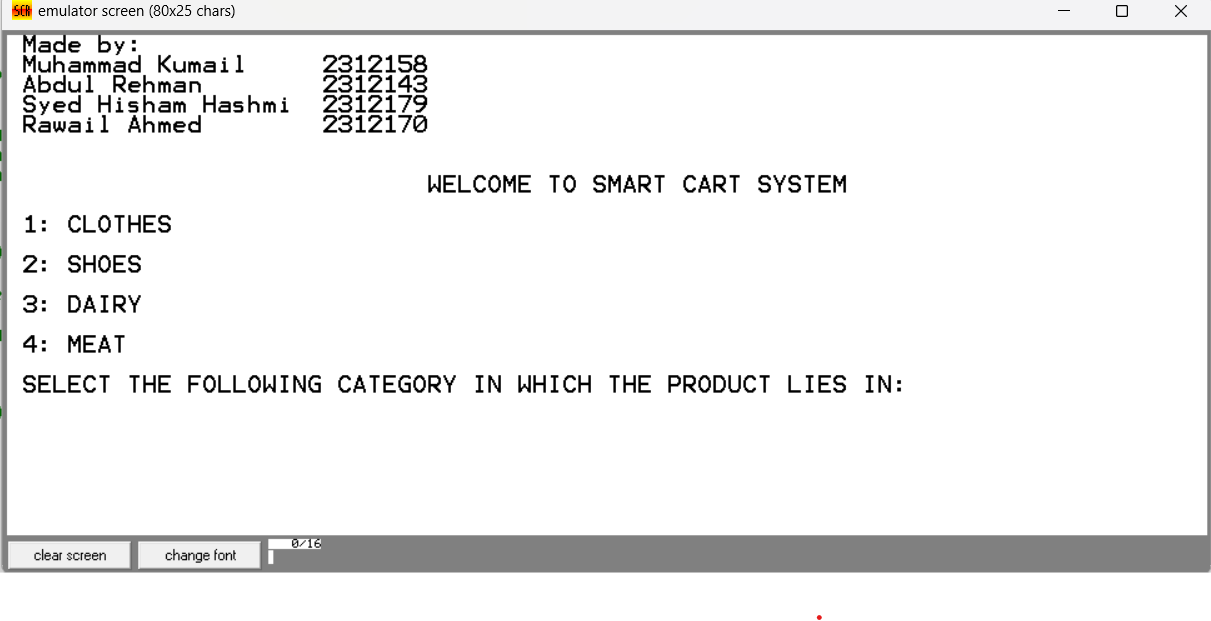
mov ah, 2h ; Function to move the cursor

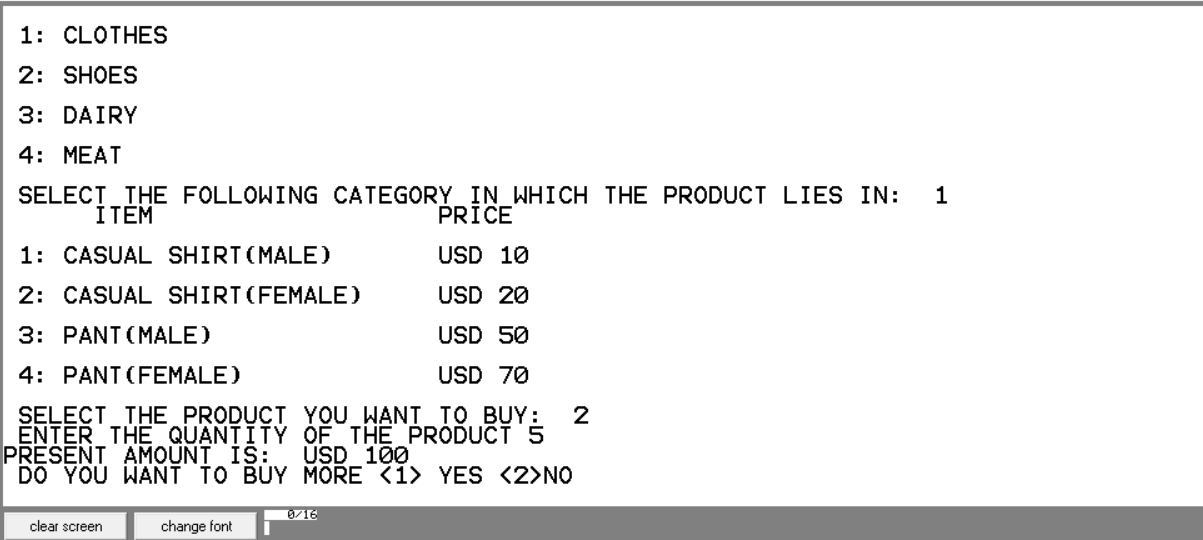
int 10h ; Call BIOS interrupt

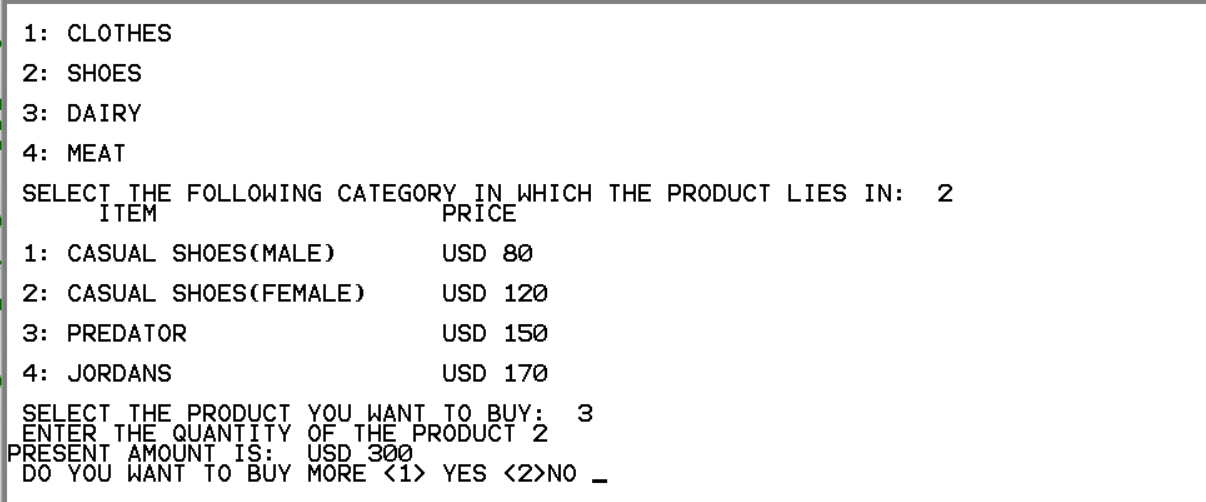
ret ; Return to the caller (main)

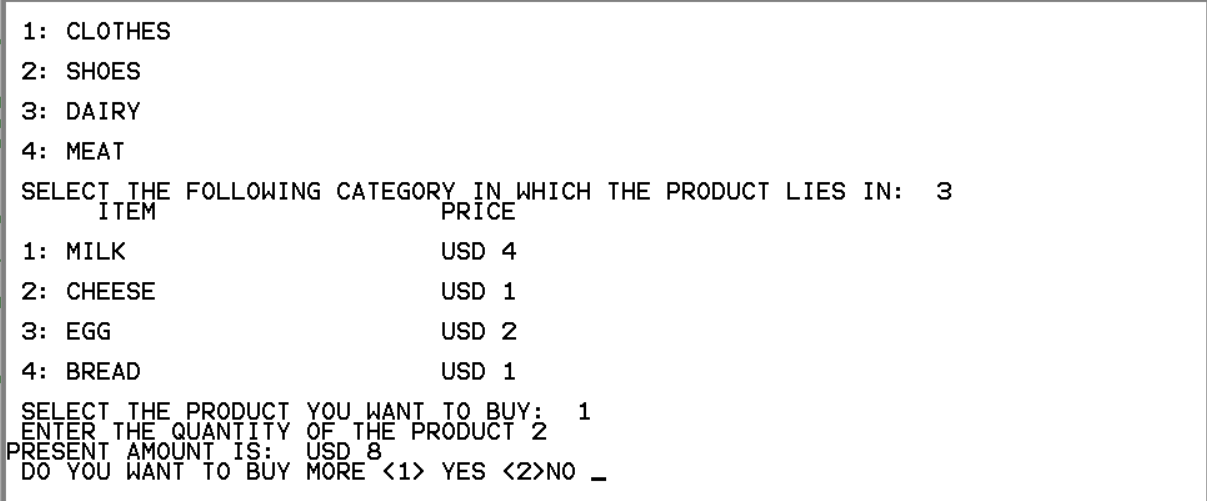
end main

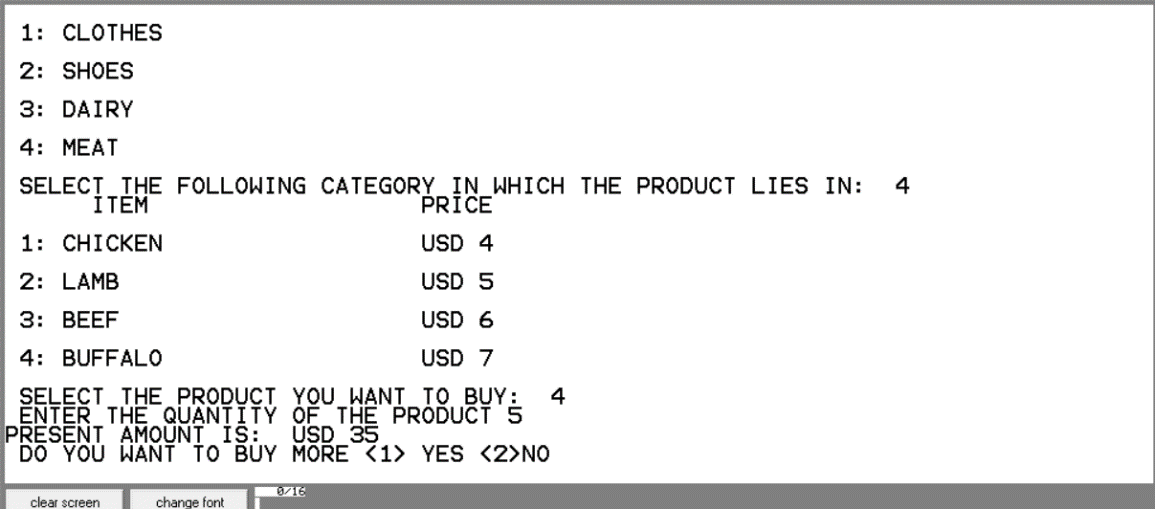
**Output**

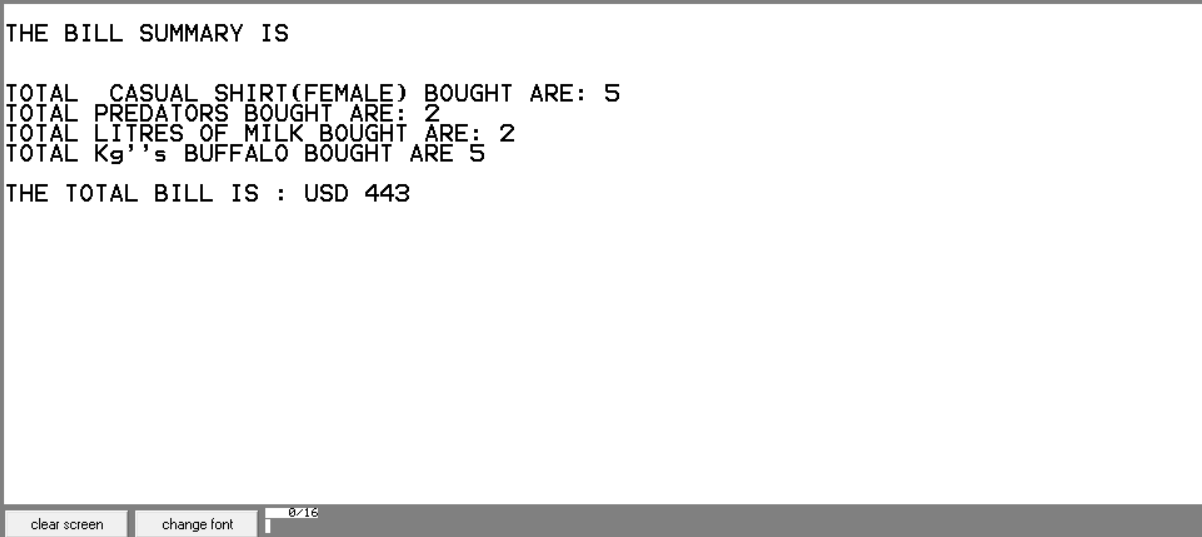
****

****

****

****

****

****

**Conclusion**

The Smart Cart System project successfully demonstrates the practical application of x86 Assembly Language in building a functional and interactive console-based application. Through programming, efficient use of registers, and effective interrupt handling, the project fulfills its objectives of providing a seamless user experience for product selection, quantity tracking, and bill calculation.

This project highlights the capability of low-level programming to implement high-level functionalities, such as menu navigation, input validation, and dynamic calculations. Despite the inherent complexities of Assembly Language, the program's clear structure and logic offer valuable insights into the efficiency and control that such a language provides.

Key achievements include the implementation of a user-friendly interface, precise error handling, and real-time updates for total cost calculations. The project also serves as an excellent example of how theoretical knowledge in Assembly Language can be applied to solve practical problems.

In conclusion, this project not only achieves its intended objectives but also provides a strong foundation for future enhancements, such as incorporating advanced features like discount mechanisms, persistent storage, and more robust input handling. It underscores the importance of low-level programming in understanding the underlying workings of computer systems while delivering functional applications.