**COAL**



**Syed Muhammad Hassan(23009105011)**

**Discipline:** BSIT

**Batch 9**

**Sir Talha**

**Sir Tayyab**

|  |
| --- |
| .model small  .stack 100h  .data  menu db '\*\*\*\*\*\*\* PARKING MANAGEMENT SYSTEM \*\*\*\*\*\*\*$', 0  menu1 db '1: Park a Vehicle$', 0  menu2 db '2: View Parking Slots$', 0  menu3 db '3: Display All Parked Vehicles$', 0  menu4 db '4: Collect Parking Fees$', 0  menu5 db '5: Delete a Vehicle$', 0  menu6 db '6: Exit Program$', 0  msg\_full db 'Parking is full!$', 0  msg\_invalid db 'Invalid input!$', 0  msg\_parked db 'Vehicle parked successfully!$', 0  msg\_no\_vehicle db 'No vehicles parked yet!$', 0  msg\_deleted db 'Vehicle deleted successfully!$', 0  msg\_fees db 'Total Parking Fee Collected: $', 0  exit\_msg db 'Thank you for using the Parking Management System!$', 0  msg\_duplicate db 'Registration number already in use!$', 0  prompt\_type db 'Enter vehicle type (R/C/B for Rickshaw/Car/Bus): $', 0  prompt\_reg db 'Enter vehicle registration number (alphanumeric and underscore only): $', 0  prompt\_slot db 'Enter slot number to delete (1-10): $', 0  slots db 10 dup(0) ; Array to track parking slots (0 = empty, 1 = occupied)  types db 10 dup(0) ; Stores vehicle types for each slot  regs db 10 dup(11 dup('$')) ; Stores registration numbers for each slot (max 10 chars + '$')  total\_fee dw 0 ; Total fees collected  fees dw 200, 300, 400 ; Parking fees: Rickshaw = 200, Car = 300, Bus = 400  input\_buffer db 11 ; Maximum length of input (10 characters + 1 for Enter)  db ? ; Actual length of input (filled by DOS)  db 11 dup(0) ; Buffer to store the input string  .code  main proc  mov ax, @data  mov ds, ax ; Initialize data segment  while\_loop:  ; Display menu  call print\_newline  mov dx, offset menu  call print\_string  call print\_newline  mov dx, offset menu1  call print\_string  call print\_newline  mov dx, offset menu2  call print\_string  call print\_newline  mov dx, offset menu3  call print\_string  call print\_newline  mov dx, offset menu4  call print\_string  call print\_newline  mov dx, offset menu5  call print\_string  call print\_newline  mov dx, offset menu6  call print\_string  call print\_newline  ; Get user input  mov ah, 1  int 21h  mov bl, al ; Store input in BL for comparison  ; Compare input and call appropriate procedure  cmp bl, '1'  je park\_vehicle  cmp bl, '2'  je view\_slots  cmp bl, '3'  je display\_vehicles  cmp bl, '4'  je collect\_fees  cmp bl, '5'  je delete\_vehicle  cmp bl, '6'  je exit\_program  ; Invalid input  mov dx, offset msg\_invalid  call print\_string  jmp while\_loop  park\_vehicle:  call park\_vehicle\_proc  jmp while\_loop  view\_slots:  call view\_slots\_proc  jmp while\_loop  display\_vehicles:  call display\_vehicles\_proc  jmp while\_loop  collect\_fees:  call collect\_fees\_proc  jmp while\_loop  delete\_vehicle:  call delete\_vehicle\_proc  jmp while\_loop  exit\_program:  mov dx, offset exit\_msg  call print\_string  mov ah, 4Ch  int 21h  main endp  park\_vehicle\_proc proc  ; Find an empty slot  mov cx, 10  lea si, slots  find\_slot:  cmp byte ptr [si], 0 ; Check if slot is empty  je park\_here ; If slot is empty, park vehicle  inc si  loop find\_slot  ; No empty slot found  mov dx, offset msg\_full  call print\_string  ret  park\_here:  ; Prompt for vehicle type  mov dx, offset prompt\_type  call print\_string  mov ah, 1  int 21h  ; Convert input to uppercase  cmp al, 'a'  jb check\_upper  cmp al, 'z'  ja check\_upper  sub al, 32 ; Convert lowercase to uppercase  check\_upper:  cmp al, 'R'  je valid\_type  cmp al, 'C'  je valid\_type  cmp al, 'B'  je valid\_type  jmp invalid\_type  valid\_type:  mov byte ptr [types + si - slots], al  ; Prompt for registration number  mov dx, offset prompt\_reg  call print\_string  lea di, regs[si - slots]  call read\_string  ; Check for duplicate registration number  call check\_duplicate\_registration  jc duplicate\_registration ; If duplicate, jump to error handling  ; Validate registration number (alphanumeric and underscore only)  call validate\_registration  jc invalid\_registration ; If invalid, jump to error handling  ; Mark slot as occupied  mov byte ptr [si], 1  ; Calculate fee based on vehicle type  cmp al, 'R'  je add\_rickshaw\_fee  cmp al, 'C'  je add\_car\_fee  cmp al, 'B'  je add\_bus\_fee  add\_rickshaw\_fee:  add total\_fee, 200  jmp confirm\_park  add\_car\_fee:  add total\_fee, 300  jmp confirm\_park  add\_bus\_fee:  add total\_fee, 400  jmp confirm\_park  invalid\_type:  mov dx, offset msg\_invalid  call print\_string  ret  invalid\_registration:  mov dx, offset msg\_invalid  call print\_string  ret  duplicate\_registration:  mov dx, offset msg\_duplicate  call print\_string  ret  confirm\_park:  mov dx, offset msg\_parked  call print\_string  ret  park\_vehicle\_proc endp  check\_duplicate\_registration proc  ; Check if the registration number already exists  lea si, regs ; Point to the start of the registration numbers array  mov cx, 10 ; Number of slots  check\_loop:  cmp si, di ; Skip comparison with the current slot  je skip\_comparison  call compare\_strings  jc duplicate\_found ; If strings match, carry flag is set  skip\_comparison:  add si, 11 ; Move to the next registration number  loop check\_loop  clc ; Clear carry flag to indicate no duplicate  ret  duplicate\_found:  stc ; Set carry flag to indicate duplicate  ret  check\_duplicate\_registration endp  compare\_strings proc  ; Compare two strings (SI and DI)  compare\_loop:  mov al, [si]  mov bl, [di]  cmp al, bl  jne strings\_differ  cmp al, '$' ; End of string  je strings\_match  inc si  inc di  jmp compare\_loop  strings\_differ:  clc ; Clear carry flag to indicate strings differ  ret  strings\_match:  stc ; Set carry flag to indicate strings match  ret  compare\_strings endp  validate\_registration proc  ; Validate registration number (alphanumeric and underscore only)  lea si, regs[si - slots] ; Point to the registration number  validate\_loop:  mov al, [si]  cmp al, '$' ; End of string  je valid\_reg  ; Check for uppercase letters (A-Z)  cmp al, 'A'  jb check\_lowercase  cmp al, 'Z'  jbe next\_char  check\_lowercase:  ; Check for lowercase letters (a-z)  cmp al, 'a'  jb check\_digit  cmp al, 'z'  jbe next\_char  check\_digit:  ; Check for digits (0-9)  cmp al, '0'  jb check\_underscore  cmp al, '9'  jbe next\_char  check\_underscore:  ; Check for underscore (\_)  cmp al, '\_'  je next\_char  next\_char:  inc si  jmp validate\_loop  invalid\_reg:  stc ; Set carry flag to indicate invalid registration  ret  valid\_reg:  clc ; Clear carry flag to indicate valid registration  ret  validate\_registration endp  view\_slots\_proc proc  ; Display parking slots  mov cx, 10  lea si, slots  mov bx, 1 ; Slot number (1 to 10)  view\_loop:  ; Display slot number and its status  cmp byte ptr [si], 0 ; Check if slot is empty  je empty\_slot  ; If occupied, display the vehicle type and registration number  mov dx, bx  add dl, '0'  call print\_char  mov dl, ':'  call print\_char  lea di, regs[si - slots]  call print\_string  mov dl, [types + si - slots]  call print\_char  call print\_newline  jmp next\_slot  empty\_slot:  ; Print empty slot message  mov dx, bx  add dl, '0'  call print\_char  mov dl, ':'  call print\_char  mov dx, offset msg\_no\_vehicle  call print\_string  call print\_newline  next\_slot:  inc si  inc bx  loop view\_loop  ret  view\_slots\_proc endp  display\_vehicles\_proc proc  ; Display all parked vehicles  mov cx, 10  lea si, slots  mov bx, 1 ; Slot counter  display\_loop:  cmp byte ptr [si], 0 ; Check if the slot is empty  je no\_vehicle  ; If vehicle is parked, display registration and type  mov dx, bx  add dl, '0'  call print\_char  mov dl, ':'  call print\_char  lea di, regs[si - slots]  call print\_string  mov dl, [types + si - slots]  call print\_char  call print\_newline  jmp next\_vehicle  no\_vehicle:  ; Skip empty slot  inc si  inc bx  loop display\_loop  ret  next\_vehicle:  inc si  inc bx  loop display\_loop  ret  display\_vehicles\_proc endp  collect\_fees\_proc proc  ; Display total collected fees  mov dx, offset msg\_fees  call print\_string  mov ax, total\_fee  call print\_number  call print\_newline  ret  collect\_fees\_proc endp  delete\_vehicle\_proc proc  ; Prompt for slot number  mov dx, offset prompt\_slot  call print\_string  mov ah, 1  int 21h  sub al, '1' ; Convert to zero-based index  lea si, slots  add si, ax  ; Check if slot is occupied  cmp byte ptr [si], 1  jne not\_found  ; Delete vehicle (empty the slot and clear vehicle data)  mov byte ptr [si], 0  lea di, regs[si - slots]  call clear\_string  mov byte ptr [types + si - slots], 0  mov dx, offset msg\_deleted  call print\_string  ret  not\_found:  mov dx, offset msg\_no\_vehicle  call print\_string  ret  delete\_vehicle\_proc endp  print\_newline proc  ; Print a new line  mov dl, 13 ; Carriage return  mov ah, 2  int 21h  mov dl, 10 ; Line feed  mov ah, 2  int 21h  ret  print\_newline endp  print\_number proc  ; Print a number in AX (total fee)  xor cx, cx ; Clear CX (used as a counter)  print\_loop:  xor dx, dx ; Clear DX before division  div word ptr [fees] ; Divide AX by the fee (for simplicity, assuming 200 as the fee)  push dx ; Push remainder (digit) to stack  inc cx ; Increment counter  cmp ax, 0 ; Check if AX is 0 (end of number)  jne print\_loop  print\_digits:  pop dx ; Pop digit from stack  add dl, '0' ; Convert to ASCII  mov ah, 2  int 21h ; Output digit  loop print\_digits  ret  print\_number endp  read\_string proc  ; Read a string into the input buffer  mov ah, 0Ah  lea dx, input\_buffer  int 21h ; Copy the input string to the registration number buffer  lea si, input\_buffer + 1 ; Point to the actual length  mov cl, [si] ; Get the length of the input  inc si ; Point to the start of the string  lea di, regs[si - slots] ; Destination buffer for registration number  copy\_loop:  mov al, [si]  mov [di], al  inc si  inc di  loop copy\_loop  mov byte ptr [di], '$' ; Add string terminator  ret  read\_string endp  print\_string proc  ; Print a string at DX  mov ah, 9  int 21h  ret  print\_string endp  print\_char proc  ; Print a character in DL  mov ah, 2  int 21h  ret  print\_char endp  clear\_string proc  ; Clear a string at DI  mov cx, 10  clear\_loop:  mov byte ptr [di], '$'  inc di  loop clear\_loop  ret  clear\_string endp  end main |