Documentação Zubber LISP

Detalhando as funções do programa

Programa Usuário:

(defun Ler_dados_viagem(): Função que lê os dados da viagem, nessa função o usuário informa seu local atual e seu local de destino.

```
(defun Ler_dados_viagem()
        (terpri)
        (princ "Local do usuario: ")
        (setq Loc_usuario (read-line))
        (terpri)
        (princ "Destino do usuario: ")
        (setq Destino_servico (read-line))
        (terpri)
)
```

(defun Ler_dados_compras(): Função que lê os dados das compras, o usuário informa de qual local ele deseja fazer uma compra. Ao informar esses dados a função irá verificar se o local informado pelo usuário é válido, caso seja válido, o usuário informa o local de entrega e o número de itens da lista. Após isso a função irá fazer verificar os itens da lista e informar o valor total da compra. Caso o local informado pelo usuário seja inválido a função pede ao usuário para informar um local de compra válido.

```
(defun Ler_dados_compras()
        (terpri)
        (princ "Local da compra: ")
        (setq Loc_compras (read-line))
        (terpri)
        (setq Pos_compra (Verif_pos Loc_compras))
        (setq Aux_compra (nth Pos_compra List_local))
        (setq Aux_tipo (Local_rua-Tipo Aux_compra))
                (when (< Aux_tipo 2)
                (princ "O local nao e comercial, digite outro: ")
                (setq Loc_compras (read-line))
```

```
        (setq Pos_compra (Verif_pos Loc_compras))
        (setq Aux_compra (nth Pos_compra List_local))
        (setq Aux_tipo (Local_rua-Tipo Aux_compra)))
(terpri)
(princ "Local de entrega: ")
(setq Destino_compra (read-line))
(terpri)
(princ "Itens vendidos no local: ")
(terpri)
(setq Tam_livraria (Tam_lista List_item_livra))
(setq Tam_supermercado (Tam_lista List_item_super))
(setq Tam_farmacia (Tam_lista List_item_farm))
(setq Tam_restaurante (Tam_lista List_item_rest))
(setq i 0)
(if (= Aux_tipo 2)
        (dotimes (i (- Tam_livraria 1))
                (setq x (nth i List_item_livra))
                (setq Aux_item_nome (Item_-Nome x))
                (setq Aux_item_preco (Item_-Preco x))
                (princ Aux_item_nome)
                (princ " --------- RS ")
                (princ Aux_item_preco)
                (terpri)
        ))
(if (= Aux_tipo 3)
        (dotimes (i (- Tam_supermercado 1))
                (setq x (nth i List_item_super))
                (setq Aux_item_nome (Item_-Nome x))
                (setq Aux_item_preco (Item_-Preco x))
                (princ Aux_item_nome)
                (princ " --------- RS ")
                (princ Aux_item_preco)
```

```
                (terpri)
        ))
(if (= Aux_tipo 4)
        (dotimes (i (- Tam_farmacia 1))
                (setq x (nth i List_item_farm))
                (setq Aux_item_nome (Item_-Nome x))
                (setq Aux_item_preco (Item_-Preco x))
                (princ Aux_item_nome)
                (princ " --------- RS ")
                (princ Aux_item_preco)
                (terpri)
        ))
(if (= Aux_tipo 5)
        (dotimes (i (- Tam_restaurante 1))
                (setq x (nth i List_item_rest))
                (setq Aux_item_nome (Item_-Nome x))
                (setq Aux_item_preco (Item_-Preco x))
                (princ Aux_item_nome)
                (princ " --------- RS ")
                (princ Aux_item_preco)
                (terpri)
        ))
(princ "Número de itens na lista de compra: ")
(setq Num_pedidos (read))
(setq Lista_compras (make-array Num_pedidos :initial-element 0))
(setq i 0)
(terpri)
(princ "Lista de compra:")
(terpri)
(setq Valor_total 0)
        (dotimes (i Num_pedidos)
                (setq x (+ i 1))
```

```
                (princ x)
                (princ ": ")
                (setq Item (read-line))
                (setf (aref Lista_compras i) Item)
                (setq Pos_preco (Verif_pos_item Item Aux_tipo))
                (if (= Aux_tipo 5)
                (setq Aux_item (nth Pos_preco List_item_rest)))
                (if (= Aux_tipo 4)
                (setq Aux_item (nth Pos_preco List_item_farm)))
                (if (= Aux_tipo 3)
                (setq Aux_item (nth Pos_preco List_item_super)))
                (if (= Aux_tipo 2)
                (setq Aux_item (nth Pos_preco List_item_livra)))
                (setq Aux_valor (Item_-preco Aux_item))
                (setq Valor_total (+ Valor_total Aux_valor))
            )
            (terpri)
            (format t "Valor da compra:  ~A" Valor_total)
)
```

(defun Pedir_servico(): Função responsável por pedir serviço, que são viagem ou compras, o usuário informa o qual serviço ele deseja, caso escolha a opção de pedir viagem, o usuário irá informar para onde deseja ir e qual a sua localização atual, esses dados são informados na função. Caso a opção escolhida seja compras, o usuário irá informar de qual local irá comprar e informar os itens da compra, caso o usuário escolha viagem irá chamar a função (defun Pedir_viagem() e caso o usuário escolha compras, irá chamar a função (defun Pedir_compra().

```
(defun Pedir_servico()
    (terpri)
    (princ "Escolha seu servico:")
    (terpri)
    (princ "1 - Viagem")
    (terpri)
    (princ "2 - Compras")
```

```
        (terpri)
        (princ "Opcao: ")
        (setq Tipo_servico (read))
        (if (= Tipo_servico 1)
                (Pedir_viagem)
        )
        (if (= Tipo_servico 2)
                (Pedir_compra)
        )
)
```

Programa Warrior:

(defstruct Destino_final): Define uma struct Destino final, que cria um ID e um local de destino.

```
(defstruct Destino_final
        ID
        Loc_destino
)
```

(defstruct Item_): Define uma struct Item, que cria um nome e um preço para cada item.

```
(defstruct Item_
      Nome
       Preco
)
```

Conjunto de funções que inicializam como nulo a lista de restaurantes, farmácias, supermercados e livrarias.

```
(setq List_item_rest (list nil))
(setq List_item_farm (list nil))
(setq List_item_super (list nil))
(setq List_item_livra (list nil))
```

(defun Ler_comercio(Nome_arquivo Opcao_) Função onde o sistema passa um nome de arquivo e uma opção que corresponde a um tipo de comercio. A função irá ler o arquivo passado com parâmetro, vai armazenar os itens linha

por linha e adiciona um item em sua lista correspondente, opção 5 adiciona restaurante, opção 4 adiciona farmácia, opção 3 adiciona supermercado, opção 2 adiciona livraria.

```
(defun Ler_comercio(Nome_arquivo Opcao_)
    (with-open-file (stream Nome_arquivo)
        (do ((line (read-line stream nil) (read-line stream nil)))
            ((null line))
            (setq Aux (split line))
            (print Aux)
            (setq Nome_item (nth 0 Aux))
            (setq Preco_item (read-from-string (nth 1 Aux)))
            (setq Item (make-Item_ :Nome Nome_item
                                   :Preco Preco_item)
            )
            (if (= Opcao_ 5)
            (setq List_item_rest (cons Item List_item_rest)))
            (if (= Opcao_ 4)
            (setq List_item_farm (cons Item List_item_farm)))
            (if (= Opcao_ 3)
            (setq List_item_super (cons Item List_item_super)))
            (if (= Opcao_ 2)
            (setq List_item_livra (cons Item List_item_livra)))
        )
    )
)
```

(defun Escrever_item(Nome_arquivo Opcao_) Função que passa um nome de arquivo e uma opção, abre o arquivo especificado, escreve o item que acabou de ser criado, adiciona na lista respectiva a opção passada com parâmetro. Opção 5 adiciona restaurante, opção 4 adiciona farmácia, opção 3 adiciona supermercado, opção 2 adiciona livraria.

```
(defun Escrever_item(Nome_arquivo Opcao_)
        (princ "Nome do item: ")
        (setq Nome_it (read-line))
        (terpri)
        (princ "Preco: ")
```

```
        (setq Preco_it (read))(terpri)
        (setq Aux (make-Item_  :Nome Nome_it
                               :PrecoPreco_it)
        )
        (if (= Opcao_ 5)
            (setq List_item_rest (cons Aux List_item_rest)))
        (if (= Opcao_ 4)
            (setq List_item_farm (cons Aux List_item_farm)))
       (if (= Opcao_ 3)
           (setq List_item_super (cons Aux List_item_super)))
       (if (= Opcao_ 2)
           (setq List_item_livra (cons Aux List_item_livra)))
       (setq Nome_aux Nome_it)
   (setq Preco_aux (write-to-string Preco_it))
   (setq linha (concatenate 'string Nome_aux ";" Preco_aux))
   (with-open-file (f Nome_arquivo :direction :output
                                   :if-exists :append
                                   :if-does-not-exist :create)
       (write-line linha f))
)
```

(defun Add_item() Função que adiciona itens, o sistema passa a opção de estabelecimento, e adiciona lista que corresponde com a opção que foi passada. Por exemplo: Caso a opção escolhida seja 5 iremos passar para a função (defun Escrever_item(Nome_arquivo Opcao_) o nome do arquivo correspondente aos itens do restaurante. Caso queria adicionar o item dentro do arquivo da lista é necessário que seja adicionado na forma padrão indicada.

```
(defun Add_item()
    (princ "Escolha o tipo de estabelecimento: ")
    (terpri)
    (princ "2 - Livraria")
    (terpri)
    (princ "3 - Supermercado")
    (terpri)
    (princ "4 - Farmacia")
```

```
    (terpri)
    (princ "5 - Restaurante")
    (terpri)
    (princ "Opcao: ")(setq Tipo (read))
    (if (= Tipo 5)
       (Escrever_item ".../zubber/arquivos/Itens_restaurantes.txt" 5))
    (if (= Tipo 4)
       (Escrever_item ".../zubber/arquivos/Itens_farmacias.txt" 4)
    (if (= Tipo 3)
       (Escrever_item ".../zubber/arquivos/Itens_supermercados.txt" 3))
   (if (= Tipo 2)
      (Escrever_item ".../zubber/arquivos/Itens_livrarias.txt" 2))
)
```

(defun Verif_pos_item(Nome Tipo) Função que verifica a posição do item dentro da lista de itens de um estabelecimento, a verificação é feita de acordo com o tipo escolhido, os tipos são: restaurante, farmácia, supermercado e livraria.

```
(defun Verif_pos_item(Nome Tipo)
      (if (= Tipo 5)
            (setq n (Tam_lista List_item_rest)))
      (if (= Tipo 4)
            (setq n (Tam_lista List_item_farm)))
     (if (= Tipo 3)
             (setq n (Tam_lista List_item_super)))
     (if (= Tipo 2)
             (setq n (Tam_lista List_item_livra)))
     (setq i 0)
     (setq n (- n 1))
     (dotimes (i n)
     (if (= Tipo 5)
            (setq x (nth i List_item_rest)))
    (if (= Tipo 4)
           (setq x (nth i List_item_farm)))
    (if (= Tipo 3)
```

```
            (setq x (nth i List_item_super)))
        (if (= Tipo 2)
            (setq x (nth i List_item_livra)))
        (setq y (Item_-Nome x))
            (if (string= y Nome)
        (return-from Verif_pos_item i)
)
)
(Add_item)
)
```

(defun Mudar_dest_final(i Aux) Função que atualiza a localização do motorista que terminou o serviço, muda o status do motorista para livre e altera sua localização.

```
(defun Mudar_dest_final(i Aux)
        (Set_status i "livre")
        (Set_local i Aux)
)
```

(defun Finalizar(ID) Função que passa o ID do motorista, verifica se esse ID está na lista de Destino_final, caso esteja irá ser armazenado a um parâmetro auxiliar o local relacionado a esse ID. Após isso a função busca dentro da lista de motoristas a posição do motorista referente ao ID, quando a posição foi achada, a função (defun Mudar_dest_final(i Aux) é chamada para passar o parâmetro “i” que é a posição na lista e o parâmetro Aux que é o novo local a ser setado para esse motorista.

```
(defun Finalizar(ID)
        (setq n (Tam_lista List_dest_final))
        (setq i 0)
        (setq n (- n 1))
        (dotimes (i n)
                (setq x (nth i List_dest_final))
                (setq Aux_id ID)
                (setq Motorista_id (Destino_final-ID x))
                (if(= Aux_id Motorista_id)
                        (setq y i)
```

```
            )
        )
        (setq z (nth y List_dest_final))
        (setq Aux (Destino_final-Loc_destino z))
        (setq List_dest_final (remove z List_dest_final))
        (setq n (Tam_lista List_motorista))
        (setq i 0)
        (setq n (- n 1))
        (dotimes (i n)
            (setq x (nth i List_motorista))
            (setq Aux_id ID)
            (setq Motorista_id (Motorista-ID x))
            (if(= Aux_id Motorista_id)
                    (Mudar_dest_final i Aux)
            )
       )
)
```

Programa Motorista:

(defstruct Motorista) Função que Cria a struct "Motorista" que possui ID, Nome, Localização, Status referentes ao motorista.

```
(defstruct Motorista
   ID
   Nome
   Localizacao
   Status
)
```

(defstruct Local_rua) Função que Cria a struct "Local_rua" que possui Nome, Tipo, Latitude, Longitude.

```
(defstruct Local_rua
    Nome
    Tipo
    Latitude
```

```
    Longitude
)
```

(defun split (string)) Função que pega a string como parâmetro separando os dados entre ";" e armazena em uma lista.

```
(defun split (string)
   (loop for i = 0 then (1+ j)
        as j = (position #\; string :start i)
        collect (subseq string i j)
        while j)
)
```

(defun Tam_lista(Lista)) Função que descobre o tamanho da lista passada como parâmetro e o retorna.

```
(defun Tam_lista(Lista)
    (setq Tam_l (length Lista))
    (return-from Tam_lista Tam_l)
)
```

(defun Muda (n Lista Elem)) Função que troca um elemento por outro em uma determinada posição na lista.

```
(defun Muda (n Lista Elem)
    (if (= n 0)
        (cons Elem (rest Lista))
        (cons (first Lista) (Muda(1- n) (rest Lista) Elem))
   )
)
```

(defun Cal_distancia(Lat_inicial Long_inicial Lat_final Long_final)) Função que calcula a distância entre o ponto inicial e o final no mapa através da latitude e longitude de ambas as posições.

```
(defun Cal_distancia(Lat_inicial Long_inicial Lat_final Long_final)
    (setq D2r 0.017453292519943295769236)
    (setq Dlong (* (- Long_final Long_inicial) D2r))
    (setq Dlat (* (- Lat_final Lat_inicial) D2r))
```

```
    (setq Temp_sin (sin (/ Dlat 2.0)))
    (setq Temp_cos (cos (* Lat_inicial D2r)))
    (setq Temp_sin2 (sin (/ Dlong 2.0)))
    (setq a (+ (* Temp_sin Temp_sin) (* (* Temp_cos Temp_cos) (* Temp_sin2
Temp_sin2))))
    (setq c (* 2.0 (atan (sqrt a) (sqrt (- 1.0 a)))))
    (return-from Cal_distancia (* 6368.1 c))
)
```

(defun Ler_local()) Função que procura em um arquivo uma lista de Local e devolve a lista.

```
(defun Ler_local()
    (setq List_local (list nil))
    (with-open-file (stream ".../zubber/arquivos/Locais.txt")
       (do ((line (read-line stream nil) (read-line stream nil)))
          ((null line))
          (setq Aux (split line))
          (print Aux)
          (setq Nome_local (nth 0 Aux))
          (setq Tipo_local (read-from-string (nth 1 Aux)))
          (setq Lat_local (read-from-string (nth 2 Aux)))
          (setq Long_local (read-from-string (nth 3 Aux)))
          (setq Local (make-Local_rua    :Nome Nome_local
                                         :Tipo Tipo_local
                                         :Latitude Lat_local
                                         :Longitude Long_local)
          )
          (setq List_local (cons Local List_local))
       )
    )
)
```

(defun Ler_motoristas()) Função que abre o arquivo lê linha por linha e armazena os dados na lista "List_motorista".

```
(defun Ler_motoristas()
    (setq List_motorista (list nil))
    (with-open-file (stream ".../zubber/arquivos/Motoristas.txt")
       (do ((line (read-line stream nil) (read-line stream nil)))
          ((null line))
          (setq Aux (split line))
          (print Aux)
          (setq ID_motorista (read-from-string (nth 0 Aux)))
          (setq Nome_motorista (nth 1 Aux))
         (setq Local_motorista (nth 2 Aux))
         (setq Status_motorista (nth 3 Aux))
         (setq Motorsita_ (make-Motorista    :ID ID_motorista
                                             :Nome Nome_motorista
                                             :Localizacao Local_motorista
                                             :Status Status_motorista)
         )
         (setq List_motorista (cons Motorsita_ List_motorista))
       )
    )
)
```

(defun Add_motorista()) Função responsável por adicionar dados na lista e no arquivo de Motoristas sendo eles ID, Nome, Localização e Status separando por ";".

```
(defun Add_motorista()
    (terpri)
    (princ "Id motorista: ")
    (setq ID (Tam_lista List_motorista))
    (princ ID)
    (terpri)
    (princ "Nome motorista: ")
    (setq Nome (read-line))
    (terpri)
```

```
    (princ "Localizacao: ")
    (setq Local_motorista (read-line))
    (terpri)
    (princ "Status: ")
    (setq Status (read-line))
    (setq Aux (make-motorista    :ID
                                 :Nome Nome
                                 :Localizacao Local_motorista
                                 :Status)
    )
    (setq List_motorista (cons Aux List_motorista))
    (setq ID_aux (write-to-string ID))
    (setq Nome_aux Nome)
    (setq Local_aux Local_motorista)
    (setq Status_aux Status)
    (setq linha (concatenate 'string ID_aux ";" Nome_aux ";" Local_aux ";" Status_aux))
    (with-open-file (f ".../zubber/arquivos/Motoristas.txt" :direction :output :if-exists
                                                            :append :if-does-not-exist :create)
    (write-line linha f))
)
```

(defun Set_status(n x)) Função que troca o status do motorista na lista.

```
(defun Set_status(n x)
    (setq z (nth n List_motorista))
    (setq z (make-Motorista    :ID (Motorista-id z)
                               :Nome (Motorista-nome z)
                               :Localizacao (Motorista-Localizacao z)
                               :Status x)
    )
    (setq List_motorista (Muda n List_motorista z))
)
```

(defun Set_local(n x)) Função que troca a localização do motorista na lista.

```
(defun Set_local(n x)
    (setq z (nth n List_motorista))
    (setq z (make-Motorista    :ID (Motorista-id z)
                               :Nome (Motorista-nome z)
                               :Localizacao x
                               :Status (Motorista-Status z))
   )
   (setq List_motorista (Muda n List_motorista z))
)
```

(defun Add_local()) Função responsável por adicionar dados na lista e no arquivo de Local sendo eles Nome, Tipo, Latitude, Longitude os separando por ";".

```
(defun Add_local()
    (terpri)
    (princ "Nome do local: ")
    (setq Nome_loc (read-line))
    (terpri)
    (princ "Tipo do local: ")
    (setq Tipo_loc (read))
    (terpri)
    (princ "Latitude: ")
    (setq Latitude_loc (read))
    (terpri)
    (princ "Longitude: ")
    (setq Longi_loc (read))
    (setq Aux (make-Local_rua    :Nome Nome_loc
                                 :Tipo Tipo_loc
                                 :Latitude Latitude_loc
                                 :Longitude Longi_loc)
    )
    (setq List_local (cons Aux List_local))
    (setq Nome_aux Nome_loc)
```

```
    (setq Tipo_aux (write-to-string Tipo_loc))
    (setq Latitude_aux (write-to-string Latitude_loc))
    (setq Longi_aux (write-to-string Longi_loc))
    (setq linha (concatenate 'string Nome_aux ";" Tipo_aux ";" Latitude_aux ";"
                                                                    Longi_aux))
    (with-open-file (f ".../zubber/arquivos/Locais.txt" :direction :output :if-exists
                                                    :append :if-does-not-exist :create)
    (write-line linha f))
)
```

(defun Verif_pos(Nome)) Função que busca se o nome passado como parâmetro é um nome de local valido na lista local se sim ele retorna o valor de sua posição.

```
(defun Verif_pos(Nome)
    (setq n (Tam_lista List_local))
    (setq i 0)
    (setq n (- n 1))
    (dotimes (i n)
        (setq x (nth i List_local))
        (setq y (Local_rua-Nome x))
        (if (string= y Nome)
            (return-from Verif_pos i)
        )
    )
    (Add_local)
)
```

(defun Verif_distancia_maior(Local_user Local_motorista i)) Função que verifica uma distância maior entre o usuário e o motorista tendo restrição de 4.5km distância entre eles, essa função só será executada se o “Identificador” validado na função “Verif_distancia” for igual a 2.

```
(defun Verif_distancia_maior(Local_user Local_motorista i)
    (setq Pos_user (Verif_pos Local_user))
    (setq Pos_motorista (Verif_pos Local_motorista))
    (setq Aux1 (nth Pos_user List_local))
```

```
    (setq Aux2 (nth Pos_motorista List_local))
    (setq Lat_usuario (Local_rua-Latitude Aux1))
    (setq Long_usuario (Local_rua-Longitude Aux1))
    (setq Lat_motorista (Local_rua-Latitude Aux2))
    (setq Long_motorista (Local_rua-Longitude Aux2))
    (setq Distancia (Cal_distancia Lat_usuario Long_usuario Lat_motorista
                                        Long_motorista))
    (if (<= Distancia 4.5)
        (Set_status i "ocupado"))
    (if (<= Distancia 4.5)
        (return-from Verif_distancia_maior 1)
    )
    (if (> Distancia 4.5)
        (return-from Verif_distancia_maior 0)
    )
)
```

(defun Verif_distancia(Local_user Local_motorista i)) Função que verifica a distância entre o usuário e o motorista tendo restrição de 2.5km distância entre eles, se for necessário uma busca mais longa o “Identificador” terá seu valor alterado possibilitando assim a chamada de função “Verif_distancia_maior”.

```
(defun Verif_distancia(Local_user Local_motorista i Identificador)
    (setq Pos_user (Verif_pos Local_user))
    (setq Pos_motorista (Verif_pos Local_motorista))
    (setq Aux1 (nth Pos_user List_local))
    (setq Aux2 (nth Pos_motorista List_local))
    (setq Lat_usuario (Local_rua-Latitude Aux1))
    (setq Long_usuario (Local_rua-Longitude Aux1))
    (setq Lat_motorista (Local_rua-Latitude Aux2))
    (setq Long_motorista (Local_rua-Longitude Aux2))
    (setq Distancia (Cal_distancia Lat_usuario Long_usuario Lat_motorista
Long_motorista))
     (if (= Identificador 2)
       (return-from Verif_distancia (Verif_distancia_maior Local_user Local_motorista i))
     )
```

```
    (if (<= Distancia 2.5)
        (Set_status i "ocupado")
    )
    (if (<= Distancia 2.5)
        (return-from Verif_distancia 1)
    )
    (if (> Distancia 2.5)
        (return-from Verif_distancia 0)
    )
)
```

(defun Verif_distancia_compra(Local_compra Local_entrega Local_motorista i))
Função que verifica a distância entre o local da compra e o local da entrega, posteriormente verifica a distância do motorista ao local da compra tendo restrição de 2.5km distância entre o motorista.

```
(defun Verif_distancia_compra(Local_compra Local_entrega Local_motorista i)
    (setq Pos_compra (Verif_pos Local_compra))
    (setq Pos_entrega (Verif_pos Local_entrega))
    (setq Pos_motorista (Verif_pos Local_motorista))
    (setq Aux1 (nth Pos_compra List_local))
    (setq Aux2 (nth Pos_entrega List_local))
    (setq Aux3 (nth Pos_motorista List_local))
    (setq Lat_compra (Local_rua-Latitude Aux1))
    (setq Long_compra (Local_rua-Longitude Aux1))
    (setq Lat_entrega (Local_rua-Latitude Aux2))
    (setq Long_entrega (Local_rua-Longitude Aux2))
    (setq Lat_motorista (Local_rua-Latitude Aux3))
    (setq Long_motorista (Local_rua-Longitude Aux3))
    (setq Distancia_compra (Cal_distancia Lat_compra Long_compra Lat_motorista
                                                        Long_motorista))
    (setq Distancia_entrega (Cal_distancia Lat_entrega Long_entrega Lat_motorista
                                                        Long_motorista))
    (if (<= Distancia_compra 2.5)
        (if (<= Distancia_entrega 5.5)
            (Set_status i "ocupado")
```

```
        )
    )
    (if (<= Distancia_compra 2.5)
        (if (<= Distancia_entrega 5.5)
            (return-from Verif_distancia_compra 1)
        )
    )
    (if (> Distancia_compra 2.5)
        (return-from Verif_distancia_compra 0)
    )
)
```

(defun Verif_livre(Opcao Local_user Local_compra)) Função que verifica a opção tipo de serviço, "Viagem"(1) ou "Compra"(2), se o serviço for de viagem será feita a verificação do status do motorista se for "livre" a função “Verif_distancia” será chamada, se toda a lista de motoristas for percorrida e não for encontrado nenhum motorista para o serviço o “Identificador” irá receber o valor 2 e o “i” que é o iterado será novamente zerado para fazer uma nova busca com uma distância maior que a anterior, no final ao encontrar um motorista valido será retornado à posição do motorista na lista, no caso do serviço de compras será feita a verificação do status do motorista se for "livre" a função “Verif_distancia_compra” será chamada para fazer o calculo da distancia e encontrar um motorista próximo ao comercio para realizar o serviço, quando encontrado será retornado à posição do motorista na lista.

```
(defun Verif_livre(Opcao Local_user Local_compra)
    (setq n (Tam_lista List_motorista))
     (setq i 0)
     (setq Tamanho_lista (- n 1))
     (setq Identificador 1)
     (if (= Opcao 1)
         (dotimes (i Tamanho_lista)
             (if (= i (- Tamanho_lista 1))
                 (setq Identificador 2)
             )
             (if (= i (- Tamanho_lista 1))
                 (setq i 0)
             )
```

```
            (setq x (nth i List_motorista))
            (setq y (Motorista-Status x))
            (setq Loc (Motorista-Localizacao x))
            (setq Parada 0)
            (if (string= y "livre")
                (setq Parada (Verif_distancia Local_user Loc i Identificador))
            )
            (if (= Parada 1)
                (setq Identificador 1)
            )
            (if (= Parada 1)
                (return-from Verif_livre i)
            )
        )
    )
    (if (= Opcao 2)
        (dotimes (i Tamanho_lista)
            (setq x (nth i List_motorista))
            (setf y (Motorista-Status x))
            (setf Loc (Motorista-Localizacao x))
            (setq Parada 0)
            (if (string= y "livre")
                (setq Parada (Verif_distancia_compra Local_compra Local_user Loc i))
            )
            (if (= Parada 1)
                (return-from Verif_livre i)
            )
        )
    )
)
```

Programa Main:

Conjunto de comandos que carregam os programas auxiliares.

(load ".../zubber/motorista.lisp")

(load ".../zubber/usuario.lisp")

(load ".../zubber/warrior.lisp")

Conjunto de funções que leem os motoristas, os locais e as lista de itens de cada tipo de estabelecimento.

(Ler_motoristas)

(Ler_local)

(Ler_comercio ".../zubber/arquivos/Itens_restaurantes.txt" 5)

(Ler_comercio ".../zubber/arquivos/Itens_farmacias.txt" 4)

(Ler_comercio ".../zubber/arquivos/Itens_supermercados.txt" 3)

(Ler_comercio ".../zubber/arquivos/Itens_livrarias.txt" 2)

(defun Pedir_viagem() Função responsável por todo o serviço de pedir viagem, onde essa função puxa os dados de todos os programas auxiliares, nessa função o usuário além de fornecer a localização atual e o destino final, recebe as informações do motorista que está mais próximo para levá-lo, os dados da viagem que são a distância e o tempo estimado até chegar ao destino, e o valor da sua corrida.

```
(defun Pedir_viagem()
    (Ler_dados_viagem)
    (setq Dest_aux Destino_servico)
    (setq Partida Loc_usuario)
    (setq Op_aux Tipo_servico)
    (setq Indice (Verif_livre Op_aux Partida ""))
    (setq Motorista_aux (nth Indice List_motorista))
    (setq Nome_aux (Motorista-Nome Motorista_aux))
    (setq Loc_atual (Motorista-Localizacao Motorista_aux))
    (setq Aux_dest_final (make-Destino_final      :ID (Motorista-ID Motorista_aux)
                                                  :Loc_destinoDest_aux)
    )
   (setq List_dest_final (cons Aux_dest_final List_dest_final))
   (princ "Dados da viagem:")
```

```
(terpri)
(terpri)
(format t "Seu/Sua motorista:  ~A" Nome_aux)
(terpri)
(setq Pos_Inicial (Verif_pos Partida))
(setq Pos_Final (Verif_pos Dest_aux))
(setq Pos_motorista (Verif_pos Loc_atual))
(setq Cal1 (nth Pos_Inicial List_local))
(setq Cal2 (nth Pos_Final List_local))
(setq Cal3 (nth Pos_motorista List_local))
(setq Lat_inicial (Local_rua-Latitude Cal1))
(setq Long_inicial (Local_rua-Longitude Cal1))
(setq Lat_final (Local_rua-Latitude Cal2))
(setq Long_final (Local_rua-Longitude Cal2))
(setq Lat_motorista (Local_rua-Latitude Cal3))
(setq Long_motorista (Local_rua-Longitude Cal3))
(setq Distancia_dest (Cal_distancia Lat_inicial Long_inicial Lat_final Long_final))
(setq Distancia_chegada (Cal_distancia Lat_inicial Long_inicial Lat_motorista
Long_motorista))
(setq Valor_base 3)
(setq Valor_km (* 1.33 Distancia_dest))
(setq Valor_total (+ Valor_base Valor_km))
(format t "Distancia ate voce ~A" Distancia_chegada)
(terpri)
(setq Tempo_chegada (/ Distancia_chegada 30))
(format t "Chegara ate voce em (min):  ~A" (* Tempo_chegada 56.528))
(terpri)
(format t "Seu destino: ~A" Dest_aux)
(terpri)
(format t "Distancia ate seu destino (Km): ~A" Distancia_dest)
(terpri)
(setq Tempo_viagem (/ Distancia_dest 30))
(format t "Chegara ao destino em (min):  ~A" (* Tempo_viagem 56.528))
(terpri)
```

```
(format t "Valor da viagem (R$):  ~A" Valor_total)
)
```

(defun Pedir_compra() Função responsável por todo o serviço de compra, onde essa função puxa os dados de todos os programas auxiliares, nessa função o usuário irá informar de qual estabelecimento ele deseja comprar, os itens que ele deseja, o valor da compra a ser efetuada e as informações do motorista que irá executar a entrega.

```
(defun Pedir_compra()
    (Ler_dados_compras)
    (setq Loc_compra_aux Loc_compras)
    (setq Loc_entrega_aux Destino_compra)
    (setq Op_aux Tipo_servico)
    (setq Indice (Verif_livre Op_aux Loc_entrega_aux Loc_compra_aux))
    (setq Motorista_aux (nth Indice List_motorista))
    (setq Nome_aux (Motorista-Nome Motorista_aux))
    (setq Loc_atual (Motorista-Localizacao Motorista_aux))
    (setq Aux_dest_final (make-Destino_final   :ID (Motorista-ID Motorista_aux)
                                               :Loc_destino  Loc_entrega_aux)
    )
    (setq List_dest_final (cons Aux_dest_final List_dest_final))
    (terpri)
    (princ "Dados da viagem:")
    (terpri)
    (terpri)
    (format t "Seu/Sua entregador(a):  ~A" Nome_aux)
    (terpri)
    (setq Pos_entrega (Verif_pos Loc_entrega_aux))
    (setq Pos_motorista (Verif_pos Loc_atual))
    (setq Cal1 (nth Pos_compra List_local))
    (setq Cal2 (nth Pos_entrega List_local))
    (setq Cal3 (nth Pos_motorista List_local))
    (setq Lat_compra (Local_rua-Latitude Cal1))
    (setq Long_compra (Local_rua-Longitude Cal1))
    (setq Lat_entrega (Local_rua-Latitude Cal2))
```

```
    (setq Long_entrega (Local_rua-Longitude Cal2))

    (setq Lat_motorista (Local_rua-Latitude Cal3))

    (setq Long_motorista (Local_rua-Longitude Cal3))

    (setq Distancia_compra (Cal_distancia Lat_motorista Long_motorista Lat_compra
Long_compra))

    (setq Distancia_entrega (Cal_distancia Lat_compra Long_compra Lat_entrega
Long_entrega))

    (format t "Estabelecimento da compra: ~A" Loc_compra_aux)

    (terpri)

    (format t "Distancia ate o estabelecimento ~A" Distancia_compra)

    (terpri)

    (setq Tempo_compra (/ Distancia_compra 30))

    (format t "Chegara no estabelecimento aproximidamente em (min):  ~A" (*
Tempo_compra 56.528))

    (terpri)

    (format t "Distancia ate voce (Km): ~A" Distancia_entrega)

    (terpri)

    (setq Tempo_entrega (/ Distancia_entrega 30))

    (format t "Chegara ate voce em (min):  ~A" (* Tempo_entrega 56.528))

)
```