Keyboard T9

UNIVERSIDADE DE ÉVORA
ESCOLA DE CIÊNCIAS E TECNOLOGIA

Relatório

Discentes:

João Condeço nº48976

Sara Amaral nº48563

Universidade de Évora

Curso: Engenharia Informática

Disciplina: Estruturas de Dados e Algoritmos 1

Docentes: Teresa Gonçalves, Lígia Rodrigues

Entregue dia 20/06/2021

Índice

1. Introdução

Este documento foi elaborado com o intuito de providenciar contexto e explicação para o trabalho prático desenvolvido a fim de conciliar os conhecimentos adquiridos na cadeira de Estruturas de Dados e Algoritmos 1 para desenvolver o programa *Keyboard T9* na linguagem de programação C.

Numa primeira instância iremos fazer uma detalhada explicação do funcionamento do programa, do ponto de vista do utilizador, bem como das variáveis e funções utilizadas para a realização do mesmo. Serão também abordadas as escolhas feitas ao longo da elaboração deste trabalho bem como das dificuldades sentidas na mesma.

É de realçar que foi tomada a decisão de implementar as funcionalidades opcionais referidas no enunciado e que serão também esclarecidas.

O software utilizado na elaboração deste trabalho foi: Apple Xcode (compilador).

2. Descrição do programa

Este programa tem como finalidade receber uma sequência de algarismos do utilizador e com base num critério predefinido baseado num teclado T9, que faz a correspondência entre cada algarismo e as possíveis letras a ele associadas, gerar a mensagem pretendida.

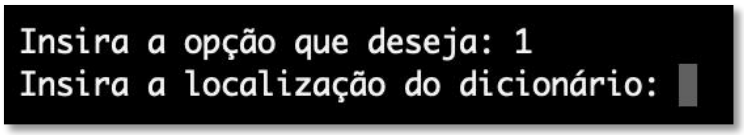
Menu principal

Quando o utilizador inicia o programa é confrontado com as seguintes opções:

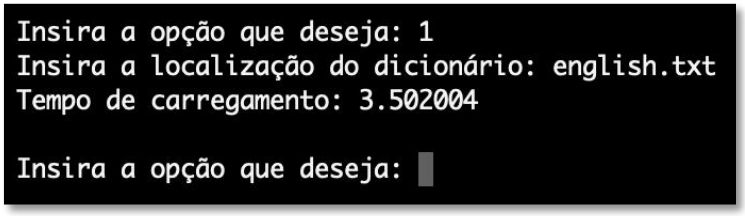
- Opção 0 (Terminar o programa) - Quando selecionada esta opção o programa termina;
- Opção 1 (Carregar dicionário) - Esta opção irá levar o utilizador a efetuar os passos necessários para carregar um dicionário no programa e obter o tempo de carregamento do mesmo. Para tal, deverá ser inserida a localização do dicionário desejado (com formatação de texto correta) e caso o ficheiro não exista o programa dará essa indicação e termina;
- Opção 2 (Informação acerca dos dígitos) – Ao selecionar esta opção será mostrada ao utilizador a mensagem ilustrada na página seguinte;
- Opção 3 (Modo Escrita) – Tal como o nome indica esta opção permite ao utilizador realizar a escrita de uma mensagem por meio da inserção de sequências de algarismos correspondentes (segundo o critério antes estabelecido) às palavras desejadas. Caso estas pertençam ao dicionário previamente selecionado serão sugeridas pelo programa as diferentes possibilidades correspondentes à sequência de algarismos introduzida, caso isto não se verifique o utilizador deverá introduzir a palavra a partir do teclado e esta será adicionada à mensagem final. Se não existir nenhum dicionário esta funcionalidade não está disponível.
- Opção 4 (Esta Mensagem) – esta mensagem mostra as opções do programa.

Dicionário

Quando selecionada a opção 1 o utilizador deparar-se-á com o seguinte:



Deverá assim ser inserida a localização do ficheiro que contém o dicionário que se pretende utilizar e, seguidamente será apresentado o tempo de carregamento do mesmo:



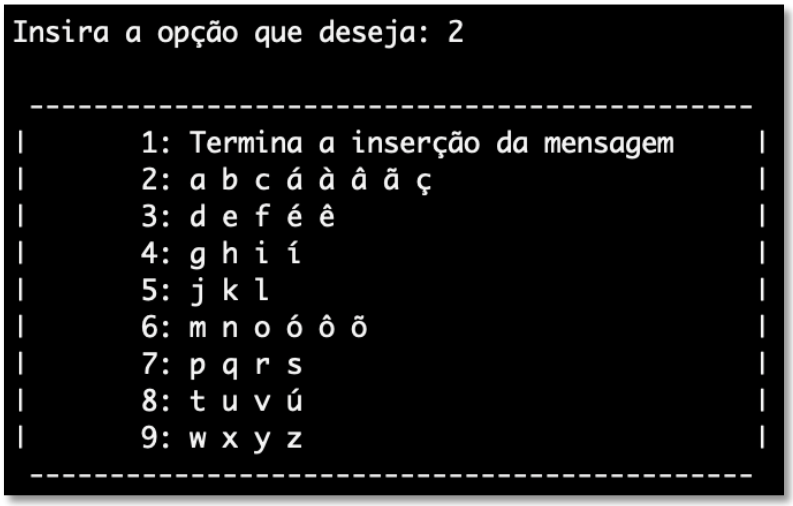
O programa retorna então ao menu principal.

Também é possível carregar o dicionário através da variável argv que recolhe o input que segue o nome do programa na sua chamada de execução a partir do terminal *(./keyboard english.txt).*

Para executar esta tarefa é chamada a função *AddDict (HashTable H, const char *argv[], int argc, int ins, int *hf, int *loaded)*.

Informação dos dígitos

Quando selecionada a opção 2 a seguinte informação é mostrada:



Inserção de texto

Selecionada a opção 3, é invocada a função *Receiver (HashsTable H)* que irá gerir a inserção de texto. Será então pedido ao utilizador para inserir uma sequência de números correspondente à primeira palavra que deseja inserir na mensagem, e a partir dela serão sugeridas palavras associadas à combinação de algarismos em causa. Caso a sugestão seja aceite a palavra será adicionada à mensagem final, caso contrário serão continuadas a ser dadas sugestões até que se esgotem as palavras do dicionário. Neste caso será pedido ao utilizador para escrever utilizando o teclado e esta palavra será posteriormente adicionada a um novo ficheiro com as palavras do dicionário já existentes (funcionalidade adicional que será abordada mais à frente neste documento).

```
Insira a opção que deseja: 3

Insira os números: 652
Sugestão: olá, aceita (s/n)? s

Insira os números: 38
Sugestão: eu, aceita (s/n)? s

Insira os números: 768
Sugestão: sou, aceita (s/n)? s

Insira os números: 33
Sugestão: fé, aceita (s/n)? n
Sugestão: de, aceita (s/n)? s

Insira os números: 46367628422
Sugestão: informática, aceita (s/n)? s

Insira os números: 1

Mensagem: olá eu sou de informática
```

```
Insira os números: 652
Sugestão: olá, aceita (s/n)? n
Sugestão: ola, aceita (s/n)? n
Não existem mais sugestões, introduza a palavra no teclado: hii

Insira os números: 1

Mensagem: hii
```

Para terminar a inserção de palavras deve ser inserido o número 1. Caso seja inserido o número 0 o programa termina.

3. Estruturas de dados utilizadas

Para este trabalho a estrutura de dados escolhida para armazenar os dicionários foi uma HashTable com hashing aberto. Esta escolha prende-se com a importância da redução ao máximo da complexidade de pesquisa devido ao elevado número de palavras contidas nos dicionários.

Foi escolhido hashing aberto ao invés de hashing fechado uma vez que várias palavras correspondem a uma mesma combinação e com esta escolha é possível atribuir o mesmo índice a uma lista de palavras com a mesma sequência de algarismos pelo que, ao fazer a sugestão de palavras é apenas necessário encontrar este índice e percorrer a lista (estrutura de dados "auxiliar" para o hashing aberto).

Após uma breve ponderação foi decidido usar como chave da função de hash a sequência de algarismos de cada palavra uma vez que diminuiria a complexidade de pesquisa pois já não seria necessário converter esta sequência, caracter a caracter, para todas as possíveis combinações de palavras e verificar a sua existência no dicionário. Ao utilizar a sequência de algarismos podemos diretamente sugerir as diferentes palavras que estão presentes no dicionário.

Foi escolhido um tamanho para a tabela de hash de 45000 para diminuir colisões, tendo sido obtida uma tabela como se pode observar:

```
i = 0 Palavras = [ findings 0 - finding's 0 -  ]
i = 1 Palavras = [ misdone 0 - discernible 0 - Lamborghini 0 -  ]
i = 2 Palavras = [ thumbnails 0 - thumbnail's 0 - inculcate 0 - gossipping 0 - Sigurd's 0 -  ]
i = 3 Palavras = [ Smithson's 0 -  ]
i = 4 Palavras = [ postpaid 0 - cartwheeled 0 -  ]
i = 5 Palavras = [ impossible 0 - deception 0 - Timothy's 0 -  ]
i = 6 Palavras = [  ]
i = 7 Palavras = [  ]
i = 8 Palavras = [ yawning 0 -  ]
i = 9 Palavras = [ portrait 0 - midyear 0 - diligence's 0 -  ]
i = 10 Palavras = [  ]
i = 11 Palavras = [ marquises 0 - marquise's 0 - impossibly 0 -  ]
i = 12 Palavras = [ visual 0 - offshore 0 - chewing 0 -  ]
i = 13 Palavras = [ collectable 0 -  ]
i = 14 Palavras = [ vistas 0 - vista's 0 - surprising 0 - richness 0 - intensively 0 -  ]
i = 15 Palavras = [ councillors 0 -  ]
i = 16 Palavras = [ winnings 0 - parochialism 0 - armistices 0 - armistice's 0 -  ]
i = 17 Palavras = [ merchanted 0 - expelled 0 - earliest 0 - driveling 0 - cerulean's 0 - Catherine's 0 -  ]
i = 18 Palavras = [ fanatics 0 - fanatic's 0 - cowpox 0 - anaesthetists 0 -  ]
i = 19 Palavras = [ phalanx 0 - intransigence 0 -  ]
i = 20 Palavras = [ rosiness's 0 - madame's 0 -  ]
i = 21 Palavras = [ detecting 0 -  ]
i = 22 Palavras = [ feinting 0 - adjured 0 -  ]
i = 23 Palavras = [ detection 0 -  ]
i = 24 Palavras = [ subcontracts 0 - subcontract's 0 -  ]
i = 25 Palavras = [  ]
i = 26 Palavras = [ keystroked 0 - hankie 0 - depreciation's 0 - adjures 0 - Salazar 0 -  ]
i = 27 Palavras = [ balminess's 0 -  ]
i = 28 Palavras = [ celerity 0 - Draco 0 -  ]
i = 29 Palavras = [ truckloads 0 - truckload's 0 - leashed 0 - drabs 0 - Hamlin 0 -  ]
i = 30 Palavras = [ universally 0 - keystrokes 0 - keystroke's 0 - chinstrap 0 -  ]
```

4. Funções do programa e variáveis utilizadas

Ficheiro k*eyboard.c*

Declaração da função: *int main(int argc, const char * argv[])*

Funcionalidade: é inicializada a *HashTable H* de tamanho *tablesz* (definido no topo do ficheiro) e caso o valor de *argc* seja maior que 1 é carregado o dicionário através dos argumentos contidos em *argv*. Caso contrário será mostrada a mensagem:

```
=> Atenção! Como não carregou um dicionário será necessário carregar um para aceder à secção 3! <=

Insira a opção que deseja:
```

E até ser inserido um dicionário não será possível aceder à inserção de texto (opção 3 do menu principal).

Através dum ciclo *while* esta função recebe a decisão (guardada na variável *option*) que o utilizador toma relativamente à funcionalidade do programa que deseja utilizar. Caso este selecione a opção 0 o ciclo é quebrado.

Na opção 1 serão utilizadas funções da biblioteca *time.h* para calcular o tempo de carregamento do ficheiro que contém o dicionário para a estrutura de dados escolhida – HashTable. O carregamento do dicionário é efetuado através da função *AddDict*.

Após a escolha da opção 2 será invocada a função *ShowInstructions().*

Por sua vez quando escolhida a opção 3 caso o valor da variável *loaded* condicionado pela função *AddDict* seja 1 (foi carregado um dicionário) é invocada a função *Receiver(H)* com a *HashTable* previamente inicializada como argumento. Caso contrário será mostrada a mensagem: "Tem de carregar um dicionário para poder executar".

Com a escolha da opção 4 será invocada a função *ShowMenu().*

Outro *outupt* conduzirá ao aparecimento da mensagem "Opção inválida, por favor tente novamente. Em caso de dúvida consulte o tutorial (insira 4)".

Uma vez quebrado o ciclo se a variável *need* condicionada pela função *Receiver* tiver um valor superior a 0 será invocada a função *upload* com a variável *has_frequencies* como argumento. Esta última é condicionada pela função *AddDict* e indica se o dicionário possui apenas as diferentes palavras ou também as suas respetivas frequências de utilização.

Por fim é chamada a função *DestroyTable* para libertar o espaço em memória previamente ocupado pela *HashTable H*.

Valor de retorno: Em condições normais esta função retorna 0.

Declaração da função: *void ShowTitle(void)*

Funcionalidade: puramente estética.

Declaração da função: *void ShowMenu(void)*

Funcionalidade: puramente estética.

Declaração da função: *void ShowInstructions(void)*

Funcionalidade: puramente estética.

Ficheiro *functions.c*

Declaração da função: *long int ToNumber(char l)*

Funcionalidade: Esta função recebe um número em forma de caracter e devolve esse mesmo número sob forma de *int*. Se l = '1' retorna o número inteiro 1.

Valor de retorno: Inteiro correspondente ao caracter.

Declaração da função: *char toLower(char l)*

Funcionalidade: Função que retorna o caracter recebido em minúsculas, assumindo que o mesmo não é um caracter especial. Por exemplo, ao receber l = 'A' retorna 'a'.

Valor de retorno: *Char* correspondente ao caracter em minúsculas.

Declaração da função: *char AccentToLower(int n)*

Funcionalidade: Esta função recebe o valor em inteiro de um caracter especial e retorna o valor em inteiro desse mesmo caracter em minúsculas.

Valor de retorno: *Int* correspondente ao caracter em minúsculas.

Declaração da função: *int SizeCounter(char *str)*

Funcionalidade: Percorre a *string* até encontrar o caracter nulo de forma a contar o número de caracteres da mesma.

Valor de retorno: *Int* correspondente ao tamanho da *string* (excluindo o caracter nulo).

Declaração da função: *int NewHash(char *str, int tablesize)*

Funcionalidade: A função recebe uma combinação de algarismos em formato de *string* bem como o tamanho da tabela de *hash* em utilização. Calcula através da função *SizeCounter* o tamanho dessa mesma *string* (guardando o mesmo na variável *size*). Seguidamente através de um ciclo *while* vai somar à variável *hashvalue* cada caracter da *string* e, à medida que avança na mesma, multiplica esta variável por 32 de modo a atribuir um "peso" a cada algarismo. A esta variável é somado o tamanho da *string*.

Valor de retorno: Resto da divisão de *hashvalue* por *tablesize*.

Declaração da função: *void print_chars(int n, char c)*

Funcionalidade: A função *print_chars* imprime o caracter desejado (variável c) as vezes indicadas (variável n). Apenas serve para auxiliar nas mensagens para o utilizador.

Declaração da função: *int InMatrix(int values[][5], int n)*

Funcionalidade: Dada a matriz *values*, esta é percorrida até encontrar o valor de n. Uma vez encontrado é retornado o valor da sua posição, caso não seja encontrado é retornado o valor -1.

Valor de retorno: Posição de n na matriz ("linha"), -1 caso este valor não pertença à mesma.

Declaração da função: *char GetNumber(char l)*

Funcionalidade: É declarado uma matriz do tipo *char* onde cada "linha" da mesma contém nas "colunas" os elementos de uma determinada tecla (teclado T9). De seguida através de dois ciclos for irá ser procurada a posição do caracter no *array* e uma vez encontrada será armazenada na variável *pos*. Terminados os ciclos será adicionado 2 à mesma de modo a obter a tecla correspondente à letra em questão.

Valor de retorno: Tecla correspondente a l sobre forma de caracter.

Declaração da função: *char *RetConvert(char *word)*

Funcionalidade: Esta função converte através da função *GetNumber* os caracteres da *string* inserida nos seus algarismos correspondentes segundo o teclado T9.

Valor de retorno: *String* com a sequência de algarismos correspondente à *string word*.

Declaração da função: *void PrintTable(HashTable H)*

Funcionalidade: Esta função auxiliar dá o output da *HashTable H*, apresentando os seus elementos e as respetivas *keys*.

Declaração da função: *void suggestions(HashTable H, char *sentence, char *in)*

Funcionalidade: Recebe uma combinação de algarismos na *string in*, seguidamente faz uso da função *NewHash* para calcular o índice da *HashTable* em que as palavras correspondentes a essa sequência de algarismos se encontram, guardando a lista que se encontra neste índice na variável do tipo *List* L. De seguida, irá percorrer essa mesma lista e sugerindo ao utilizador as palavras que aí se encontram. Quando o utilizador aceitar uma das sugestões o ciclo termina. Uma vez que existem palavras nesta lista que não correspondem à combinação de algarismos em causa (facto inevitável aquando da utilização de Hashing aberto) é necessário testar a condição que verifica se a sequência de algarismos da palavra a sugerir é igual à inserida pelo utilizador (é utilizada a função *strcmp* (biblioteca *string.h*)). Após o final do ciclo se uma das

opções foi seleciona através da função *strcat* esta é adicionada à *string* que contém a frase final (*sentence*), caso contrário será pedido ao utilizador para introduzir do teclado a palavra desejada. Esta será de igual modo adicionada à variável *sentence* e neste caso será incrementada a variável *up.posnew*. Para além disto será também guardada na variável *up.new* que irá armazenar todas as palavras desconhecidas que devem ser adicionadas ao dicionário no final do programa.

Declaração da função: *char *normalize(char *str)*

Funcionalidade: Esta função tem como objetivo tornar os caracteres especiais no seu "equivalente" caracter normal. Para isto é criado uma matriz onde cada "linha" contém as diferentes "variações" de um caracter (o seu valor do código ASCII em decimal). De seguida é criado um *array* com os diferentes caracteres "normais" correspondentes aos possíveis caracteres especiais. A *string* é percorrida através de um ciclo *while*, ignorando o caracter ' ' ' (apóstrofe) e verificando se se trata de um caracter especial. Uma vez que um caracter especial ocupa duas posições dum array e ambas possuem valor negativo, quando é encontrado um valor negativo avança-se uma posição no array e procede-se para a sua conversão (uma vez que o primeiro valor é igual a -61 independentemente de qual seja o caracter). Posteriormente é utilizada a função *InMatrix* para obter a posição do caracter em questão na matriz *values*, sendo adicionado à palavra final (*string new*) o caracter "normalizado".

Caso se trate de um caracter normal este será adicionado à palavra final. Antes de qualquer procedimento são chamadas as funções *AccentToLower* e *toLower* a fim de obter o mesmo caracter mas em minúsculas (para facilitar o funcionamento do programa).

Valor de retorno: *String* "normalizada".

Declaração da função: *char *NewName(char *name)*

Funcionalidade: Esta função recebe o nome de um ficheiro e, através dum ciclo *for*, copia caracter a caracter da variável *name* para a variável *new* tudo exceto a terminação ".txt" (para tal o ciclo vai de 0 até *strlen(name) – 4*, sendo 4 o tamanho de ".txt"). De seguida, através da função *strcat* (biblioteca *string.h*) é adicionado ao fim da nova *string* "-updated.txt".

Valor de retorno: *String* com o nome do ficheiro mais ""-updated.txt".

Declaração da função: *void upload(int hf)*

Funcionalidade: Esta função baseia-se na variável *hf* (verifica se o dicionário contém as frequências de utilização de cada palavra) para definir o tipo de ficheiro que vai gerar. Inicialmente irá copiar o conteúdo do dicionário carregado para um novo ficheiro e de seguida adiciona as palavras novas guardadas na variável global *up.new*. Caso a variável *hf* possua o valor 1 irá ser registado no novo ficheiro a frequência de utilização das palavras por adicionar (este valor será 0 tal como indicado no enunciado). Caso contrário não serão introduzidas no novo ficheiro quaisquer frequências.

Declaração da função: *int Value(char *numbers)*

Funcionalidade: Esta função assume que a variável *word* é constituída por algarismos e retorna o valor em inteiro desses mesmos algarismos. Por exemplo ao receber a *string* "1001" é retornado o valor em inteiro 1001.

Valor de retorno: Inteiro correspondente à sequência de algarismos da *string* recebida.

Declaração da função: *char *TInput(char input[50], int *pf)*

Funcionalidade: Esta função tem como finalidade separar o input lido do ficheiro com o dicionário numa palavra e na sua respetiva frequência. A palavra será retornada na variável *threated* e o valor da frequência será guardado na variável apontada por *pf.*

Valor de retorno: Palavra em *threated*.

Declaração da função: *HashTable AddDict(HashTable H, const char *argv[], int argc, int ins, int *hf, int *loaded)*

Funcionalidade: Esta função começa por verificar statravés de argc e ins se é necessário pedir ao utilizador a localização do dicionário, caso seja necessário será pedido e armazenado na variável *location* e na variável *up.name* (mais tarde esta última será utilizada na função *upload*). Caso contrário será copiada a localização armazenada em *argv* para as mesmas variáveis mencionadas. Seguidamente tentar-se-á abrir o ficheiro, caso não seja possível o programa termina, caso contrário irá ter início um ciclo *while* que é quebrado quando chegar ao fim do ficheiro (EOF). Dentro deste ciclo será usada a função *TInput* e, palavra por palavra serão adicionadas à *HashTable* através da função *Insert*. Caso o dicionário contenha as frequências de utilização das palavras a variável *hf* é atualizada e passa a possuir o valor 1. Caso contrário a variável *freq* continua com valor 0. Após tudo isto, a variável *loaded* passar a ater o valor 1, indicando que o dicionário foi carregado com sucesso.

Valor de retorno: HashTable atualizada.

Declaração da função: *int Receiver(HashTable H)*

Funcionalidade: Armazena na variável global *up.posnew* o valor 0 a fim de calcular quantas palavras novas são inseridas ao longo do processo que se segue. Através de um ciclo *while* recebe o input correspondente a diferentes sequências de algarismos decifrados pela função *sugestions*. Este ciclo será quebrado quando for inserido o número 1 e termina o programa se for inserido o número 0. Uma vez quebrado o ciclo irá mostrar a mensagem constituída pelas várias combinações inseridas.

Valor de retorno: Número de palavras novas inseridas pelo utilizador que não pertenciam ao dicionário.

Ficheiro *hashsep.c*

Neste ficheiro encontram-se as funções relativas à implementação de Hashing aberto dadas pelos docentes da disciplina. De modo a garantir a compatibilidade com o programa foram feitas algumas alterações mencionadas ao longo deste relatório.

Declaração da função: *int Hash(char *str, int tablesize)*

Funcionalidade: Esta função garante que a *string* recebida possui o formato correto para ser passada como argumento à função *NewHash*. Assim faz uso das funções *RetConvert* e *normalize*.

Valor de retorno: Valor de retorno da função *NewHash*.

Declaração da função: *void Insert(char *Key, int freq, HashTable H)*

Funcionalidade: Esta função foi modificada de modo a ter em conta a frequência de utilização de cada palavra do dicionário. Para tal foi adicionada ao tipo composto *ListNode* a variável *Freq* que armazena esta informação. De modo a que a função *suggestions* possa fazer, sem quaisquer alterações adicionais, a sugestão de palavras por ordem decrescente de frequência de utilização é necessário fazer a inserção com algumas condições extra.

É guardado na variável P do tipo *Position* o valor de L onde se inicia a lista correspondente a cada índice. De modo a garantir as condições mencionadas anteriormente é necessário avançar na lista até encontrar um elemento com frequência de utilização menor que a palavra que se pretende inserir ou até se chegar ao fim da mesma, para isso é utilizado um ciclo *while*. Quando é encontrada a posição ideal de inserção basta estabelecer as ligações corretas entre as listas.

5. Dificuldades

Dificuldades: cena de ler frequências e 1 so função de hash

Ao longo da elaboração do trabalho deparámo-nos com algumas dificuldades, tendo sido as seguintes as mais problemáticas:

- Leitura do ficheiro de dicionário com frequências: ao fazer *fscanf* de cada linha do documento não era feita a separação da palavra e da sua frequência. A variável *freq* ficava sempre com o valor 0 e a frequência (bem como a vírgula) eram copiadas para a *string* que deveria conter apenas a palavra. A única maneira encontrada para solucionar este problema foi a criação da função *Tinput*;
- Função de hash: aquando da modificação da função de hash tentámos condensar ambas as funções (*Hash* e *NewHash*) numa só, porém, existiam colisões de palavras com tamanhos diferentes pelo que resolvemos criar duas funções, mesmo após criadas ao tentar condensar existiam *warnings* de compilação pelo que resolvemos deixar ambas a existir em separado;
- Tal como mencionado foram feitas mudanças ao ficheiro com a implementação do Hashing aberto: mudança do *ElementType* para *char* pois estava a criar problemas e mudança das funções *Hash* e *Insert* como mencionado anteriormente (bem como adição do parâmentro *Freq* ao *ListNode*);
- Um outro problema foi a ligação entre os ficheiros (.h e .c) pois eram gerados símbolos duplicados para a arquitetura x86_64 (*error: duplicate symbols for architecture x86_64*).

6. Conclusão

Em suma, consideramos que adquirimos bastante experiência aquando da realização do trabalho especialmente em relação às estruturas de dados HashTables e listas ligadas. Fomos também capazes de solucionar todos os problemas que surgiram e garantir a funcionalidade total do ficheiro e inexistência de *warnings*.

Assim, consideramos que a decisão de realizar este trabalho terá sido a adequada.