/\*

\* File: main.c

\* Author: F. Mário Martins - LI3 - 2019

\*

\*/

#include <stdio.h>

#include <stdlib.h>

//#define \_SVID\_SOURCE // Este DEFINE permite suprimir warnings de strdup

#define \_GNU\_SOURCE

#include <string.h>

#define CAMPOSVENDA 7

struct venda {

char\* codProd;

char\* codCli;

double precoUnit;

int quantidade;

char\* tipo;

int mes;

int filial;

};

typedef struct venda VendaBasica;

typedef struct venda\* Venda;

//---------------- FUNÇÕES AUXILIARES ----------------------

/\* FUNÇÃO QUE INSERE UMA STRING NUM ARRAY DINÂMICO DE STRINGS

\* REALOCA PARA INSERIR MAIS UMA STRING

\* O TAMANHO DO ARRAY VAZIO É 1 PORQUE TEM 1 NULL; O TAMANHO DO

\* ARRAY É SEMPRE LASTINDEX+1;

\* AO REALOCAR TEMOS QUE GARANTIR QUE O TAMANHO DO NOVO ARRAY É

\* LASTINDEX+2:

\* EM CASO DE ERRO DEVOLVE O ARRAY ORIGINAL

\*/

char\*\* insere\_CodProd(char\*\* arrayProds, const char \*codProd) {

int lastIndex = 0;

// verifica qual o 1º disponível

while (arrayProds[lastIndex] != NULL)

lastIndex++;

// realoca o array

char\*\* newArrayProds = (char\*\*) realloc(arrayProds, (lastIndex+2) \* sizeof(char\*));

if (!newArrayProds) return arrayProds;

char\* novoProd = strdup(codProd);

if (!novoProd) return newArrayProds;

newArrayProds[lastIndex] = novoProd;

lastIndex++;

newArrayProds[lastIndex] = NULL;

return newArrayProds;

}

void printArrayDyn (char\*\* array) {

char\* str; int i=0;

if(array[0] == NULL) printf("ARRAY VAZIO !!\n");

else {

while ((str = array[i]) != NULL) {

printf("%s\n", str);

i++;

}

}

}

/\* Função que dada uma linha de venda cria um array campos[] com os 7 tokens

\* Versão normal em que o array vazio é passado como parâmetro e os tokens inseridos;

\*/

void tokenizeLinhaVenda(char\* vendaRaw, char\* campos[]) {

int index = 0;

char\* token = strtok(vendaRaw," ");

while(!(token == NULL)) {

campos[index] = strdup(token);

// printf(" %s\n", token);

token = strtok(NULL," ");

index++;

}

}

/\* Função que dada uma linha de venda cria um array dinâmico

\* onde são inseridos os 7 tokens

\* Versão com alocação dinâmica

\*/

char\*\* tokenizeLinhaVendaDyn(char\* vendaRaw) {

int index = 0;

char\*\* campos = (char\*\*) malloc(CAMPOSVENDA \* sizeof(char\*)); // dimensão vertical

char\* token = strtok(vendaRaw," ");

while(!(token == NULL)) {

campos[index] = strdup(token);

// printf(" %s\n", token);

token = strtok(NULL," ");

index++;

}

return campos;

}

VendaBasica mkVendaBasica(char\* linhaVenda) {

char\* campos[CAMPOSVENDA];

VendaBasica vendaAux;

int index = 0;

char\* token = strtok(linhaVenda," ");

while(!(token == NULL)) {

campos[index] = strdup(token);

printf("%s\n", token);

token = strtok(NULL," ");

index++;

}

vendaAux.codProd = strdup(campos[0]);

vendaAux.codCli = strdup(campos[3]);

vendaAux.precoUnit = atof(campos[1]);

vendaAux.quantidade = atoi(campos[2]);

vendaAux.tipo = campos[4];

vendaAux.mes = atoi(campos[5]);

vendaAux.filial = atoi(campos[6]);

return vendaAux;

}

Venda mkVenda(char\* linhaVenda) {

char\* campos[CAMPOSVENDA];

Venda vendaAux;

vendaAux = (Venda) malloc(sizeof(struct venda));

int index = 0;

char\* token = strtok(linhaVenda," ");

while(!(token == NULL)) {

campos[index] = strdup(token);

printf("%s\n", token);

token = strtok(NULL," ");

index++;

}

vendaAux -> codProd = strdup(campos[0]);

vendaAux -> codCli = strdup(campos[3]);

vendaAux -> precoUnit = atof(campos[1]);

vendaAux -> quantidade = atoi(campos[2]);

vendaAux -> tipo = campos[4];

vendaAux -> mes = atoi(campos[5]);

vendaAux ->filial = atoi(campos[6]);

return vendaAux;

}

Venda mkVendaStruct(char\* linhaVenda) {

char\*\* campos;

Venda vendaAux;

vendaAux = (Venda) malloc(sizeof(struct venda));

campos = tokenizeLinhaVendaDyn(linhaVenda);

vendaAux -> codProd = strdup(campos[0]);

vendaAux -> codCli = strdup(campos[3]);

vendaAux -> precoUnit = atof(campos[1]);

vendaAux -> quantidade = atoi(campos[2]);

vendaAux -> tipo = campos[4];

vendaAux -> mes = atoi(campos[5]);

vendaAux ->filial = atoi(campos[6]);

return vendaAux;

}

void printVendaBasica(VendaBasica vendaAux) {

printf("Produto %s - ", vendaAux.codProd);

printf("Cliente: %s ", vendaAux.codCli);

printf("Preco: %f ", vendaAux.precoUnit);

printf("Quant: %d ", vendaAux.quantidade);

printf("Tipo: %s ", vendaAux.tipo);

printf("Mes: %d ", vendaAux.mes);

printf("Filial: %d\n ", vendaAux.filial);

printf("Total da Venda = %f\n", vendaAux.precoUnit \* vendaAux.quantidade);

}

void printVenda(Venda vendaAux) {

printf("Produto %s - ", vendaAux -> codProd);

printf("Cliente: %s ", vendaAux -> codCli);

printf("Preco: %f ", vendaAux -> precoUnit);

printf("Quant: %d ", vendaAux -> quantidade);

printf("Tipo: %s ", vendaAux -> tipo);

printf("Mes: %d ", vendaAux -> mes);

printf("Filial: %d\n ", vendaAux -> filial);

printf("Total da Venda = %f\n", vendaAux->precoUnit \* vendaAux->quantidade);

}

//----------------- TESTES ------------------------------------

int main(int argc, char\*\* argv) {

//--- Testa a função TOKENIZER (versão 1) ----------------------

char\* linha0 = strdup("XY1912 121.5 15 Z3249 N 11 1");

char\* campos[CAMPOSVENDA];

tokenizeLinhaVenda(linha0, campos);

printf("Prod %s - Cliente %s\n", campos[0], campos[3]);

printf("----------------------\n");

//--- Testa a função TOKENIZER (versão 2) ----------------------

linha0 = strdup("AB9108 76.5 18 A6649 P 4 2");

char\*\* campos1;

campos1 = tokenizeLinhaVendaDyn(linha0);

printf("Prod %s - Cliente %s\n", campos1[0], campos1[3]);

printf("----------------------\n");

//--- Testa o uso de uma STRUCT básica ----------------------

char\* linha1 = strdup("KY9812 12.35 10 F4499 N 7 3");

VendaBasica vb = mkVendaBasica(linha1);

printVendaBasica(vb);

//--- Testa o uso de uma STRUCT acedida por \* (tal como deve ser codificado ---

char\* linha2 = strdup("KK1832 102.5 6 W1199 P 10 1");

Venda v = mkVenda(linha2);

printVenda(v);

//--- Testa mkVendaStruct que chama tokenizeLinhaVendaDyn()

char\* linha3 = strdup("VS8866 2.5 634 G1098 N 5 2");

Venda v1 = mkVendaStruct(linha3);

printVenda(v1);

//---------------------------------------------------------------------

char\*\* arrayProdutos = (char\*\*) malloc(1 \* sizeof(char\*));

arrayProdutos[0] = NULL;

printArrayDyn(arrayProdutos);

arrayProdutos = insere\_CodProd(arrayProdutos, "KK1234");

arrayProdutos = insere\_CodProd(arrayProdutos, "PK2345");

arrayProdutos = insere\_CodProd(arrayProdutos, "RS9912");

printArrayDyn(arrayProdutos);

return (EXIT\_SUCCESS);

}