3º Teste de Princípios de Programação Procedimental
16 de junho de 2023
Duração 2 horas

Departamento de Engenharia Informática - FCTUC - Universidade de Coimbra
Licenciatura em Engenharia Informática

Este teste é com consulta, mas apenas de recursos em papel. É proibido o uso de qualquer aparelho eletrónico (smartwatches, telemóveis, tablets, computadores, etc.). Quem tiver trazido esses aparelhos deve desligá-los e colocá-los no chão, não podendo mexer neles durante a prova. A violação desta regra, assim como qualquer tentativa de comunicar com qualquer pessoa para além dos docentes que estão a vigiá-la, pode levar à anulação da prova.

Nota: Preencha com o nome e nº completos e responda nos espaços reservados para o efeito. **Duração total**: 2h.
Nome:_____ Número:_____

1. O programa seguinte pretende guardar nomes de pessoas, constituídos apenas por uma palavra cada, e as respetivas datas de nascimento, num array. Começa por perguntar ao utilizador quantas pessoas pretende registar, solicitando, de seguida, os nomes e respetivas datas de nascimento, no formato '`nome dia/mês/ano`'. Deverá ser utilizada apenas a memória estritamente necessária para guardar os dados das pessoas.

Exemplo:
```
Quantas pessoas pretende registar: 2
Nome data de nascimento (nome dia/mes/ano): Nuno 12/12/2000
Nome data de nascimento (nome dia/mes/ano): Maria 10/10/2001
```

O programa seguinte pretende corresponder ao solicitado:
```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

struct data {
    int dia, mes, ano;
};

struct pessoa {
    char * nome;
    struct data * data_nasc;
};
```

Identifique e corrija os **quatro** erros ou omissões existentes no código seguinte, identificando cada linha a corrigir e a respetiva correção. (**Nota:** o número de linhas é fixo!)

```
11: void add_pessoa (int pos, char * nome, struct data d, struct pessoa * pessoas []) {
12:     pessoas[pos].nome = (char*)malloc(sizeof(char));
13:     pessoas[pos].nome = nome;
14:     pessoas[pos].data_nasc = (struct data *)malloc (sizeof (struct data));
15:     pessoas[pos].data_nasc = d;
16: }
17: int main (){
18:     struct data dt; char nome [50]; int qt;
19:     printf ("Quantas pessoas pretende registar: "); scanf (" %d", &qt);
20:     struct pessoa *pessoas = (struct pessoa *)malloc((qt)*sizeof(struct pessoa));
21:     for (int i=0; i<qt; i++){
22:         printf ("Nome data de nascimento (nome dia/mes/ano): ");
23:         scanf (" %s %d/ %d/ %d", nome, &dt.dia, &dt.mes, &dt.ano);
24:         add_pessoa (i, nome, dt, pessoas);
25:     }
26:     /* TODO fazer o resto do programa */
27:     return 0;
28: }
```

Nome:_____ Número:_____

2. Considere o seguinte programa, em que a estrutura `aluno` tem o propósito de guardar informação sobre um aluno de um curso (nome, número e disciplinas em que está inscrito). Complete o seguinte código cujo objetivo é corrigir o nome de uma disciplina de um determinado aluno guardado no ficheiro **dados.bin**. Considere que no ficheiro **dados.bin** foram sequencialmente guardados os dados de vários alunos com a função **fwrite**.

```
#include <stdio.h>
#include <string.h>

typedef struct aluno {
    char nome [50];
    int numero;
    char disciplinas[5][10];
} aluno;

void alteraDisciplina (FILE * f, ____________, int i, int n){
        aluno a;
        fseek(f, _________________, SEEK_SET);
        fread (________, sizeof (struct aluno), 1, f);
        fseek(f, ____________________________, _______________);
        strcpy(_____________________, d);
        fwrite(&a, ___________________, _____________, f);
}

int main(){
    FILE * f = fopen("dados.bin", ____);
    if (f!=NULL){
       alteraDisciplina (f, "novoNome", 3, 2); // alterou a disciplina 3 do aluno 2
       ____________ (f);
    }
    return 0;
}
```

Nome:_____ Número:_____

3. Considere o seguinte programa que usa uma pilha para armazenar uma string. Cada nó da pilha é um carater. Existem as seguintes operações na pilha: *is_empty*, *push*, *pop* e *delete*.

```
typedef struct stack{
    char c;
    struct stack *next;
}type_stack;
```

```
int is_empty(type_stack **s);
void push(type_stack **s, char c);
char pop(type_stack **s); // -1 se pilha vazia
void delete(type_stack **s);
```

Existem três tipos de operações sobre a string definidas no array de inteiros *ops*: imprimir a string (0), adicionar um carater (1) e remover um carater (2).
Quando a pilha estiver vazia durante a remoção, é necessário imprimir a mensagem "Stack empty!".
Não existem valores inválidos no array *ops*.

Implemente as funções ***string_to_stack*** **e** ***string_operations***. Assuma que o carater a inserir é dado pela função ***"int getchar()"*** da biblioteca "***stdio.h***".

```
// Insere uma string na pilha
void string_to_stack(type_stack **s, char *str);

// Executa as instruções definidas no array ops
void string_operations(type_stack **s, int *ops, size_t size_ops);
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char **argv) {
    char str[] = "abcde";
    int ops[] = {0,1,0,1,0,2,2,0};
    size_t size_ops = sizeof(ops)/sizeof(int);
    type_stack *s = NULL;

    string_to_stack(&s, str);
    string_operations(&s, ops, size_ops);
    delete(&s);

    return 0;
}
```

Exemplo:

```
str → "abcde"
ops → {0,1,0,1,0,2,2,0}
```

Current string: edcba
Added: a
Current string: aedcba
Added: a
Current string: aaedcba
Removed: a
Removed: a
Current string: edcba

Nome:_____ Número:_____

4. Considere um contexto onde são usadas duas listas ligadas para guardar informação relativa a clientes e a compras realizadas por estes. As estruturas de dados são as seguintes:

```
#define MAX_STR 101

typedef struct no_lista_cliente {
    int id;
    char nome[MAX_STR];
    struct no_lista_cliente *proximo;
} no_lista_cliente;

typedef struct no_lista_compra {
    int id_compra, id_cliente;
    double valor;
    struct no_lista_compra *proximo;
} no_lista_compra;
```

Implemente a função ***maior_compra*** que recebe estas duas listas e devolve uma **struct** com a informação da maior compra realizada. A **struct** referida é a seguinte:

```
typedef struct {
    int id_cliente, id_compra;
    char nome_cliente[MAX_STR];
    double valor;
} info_compra;
```

O protótipo da função é o seguinte:

info_compra maior_compra(no_lista_cliente **clientes, no_lista_compra **compras);

Caso não existam compras, a função deve devolver os campos id_cliente, id_compra e valor a -1, e o campo nome_cliente com uma string vazia. É garantido que existe coerência de informação entre as duas listas. Nomeadamente, se existir uma compra com um dado ID de cliente, é garantido que existe um nó na lista de clientes com esse ID.