Instituto Superior de Engenharia de Lisboa

Programação II

2023/24 – 1.º semestre letivo

Exame de Época Especial

2024.02.20

Este exame é cotado para 20 valores e tem duração de **2 horas e 30 minutos**.

Em todos os exercícios que usam **alojamento dinâmico**, assuma, por simplificação, que o mesmo é **sempre bem sucedido**, não ocorrendo falta de memória de *heap*.

Nota importante: Valoriza-se a escrita de código que inclua **comentários esclarecedores** da implementação seguida e que contenha **indentação legível**.

1. [2 valores]

Pretende-se identificar sequências de bits a 1 numa palavra em binário natural e verificar se alguma sequência atinge uma determinada dimensão.

Escreva a função

```
int checkOnesSequence( unsigned long data, int minSize );
```

que verifica se o parâmetro `data` contém alguma sequência de bits consecutivos com o valor 1, cuja dimensão seja, pelo menos, a indicada em `minSize`. Em caso afirmativo, retorna 1; caso contrário, retorna 0. Com vista à portabilidade do código, se necessitar de identificar a dimensão da palavra, deve usar o operador `sizeof` e a macro `CHAR_BIT` definida em `limits.h`.

2. [4 valores]

Considere o troço de código constante da caixa seguinte. Na linha 1 daquele troço de código deverá inserir o algarismo das unidades do seu número de aluno no local indicado. Por exemplo: se o seu número de aluno fosse o 12345, aquela linha ficaria assim: `#define ALG 5`

Nota importante: a utilização do algarismo errado penalizará significativamente a avaliação da alínea b) deste grupo.

```
1  #define ALG algarismo_das_unidades_do_seu_número_de_aluno
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5  int show(int *e, void * p){
6    printf("%d ", *e);
7    return 1;
8  }
9  int f2(const int * e) {
10   return (*e == ALG);
11 }
12 int chg(int * n, int * lim){
13   int r = (*lim > 4);
14   *n = r ? *n * *n : -*n;
15   return r;
16 }
```

```
17 int f1(int a[], size_t size, int (*cond)(const int * e),
18         int (*act)(int *data, void *context), void * context) {
19   size_t i, s=0;
20   for(i = 0; i < size; i++)
21     if(cond == NULL || cond(a+i))
22       s += act(a+i, context);
23   return s;
24 }
```

a) [1] No troço de programa seguinte consta um exemplo de utilização da função `f1`.

```
int *a, option=ALG, r;
(…) // Preenchimento do array dinâmico a.
size_t size=20;
r = f1(a, size, NULL, show, NULL);
```

Admita que o conteúdo do *array* **a** antes da invocação da função `f1` é o seguinte:

-4	3	-1	-3	0	4	-7	-8	-6	5	-9	7	-5	8	0	6	2	1	9	-2

Indique, justificando, qual é o *standard output* produzido pela execução deste troço de código e apresente, justificando, o conteúdo da variável **r**.

b) [1,5] No troço de programa seguinte consta outro exemplo de utilização da função `f1`.

```
int *a, option=ALG, r;
(…) // Preenchimento do array dinâmico a.
size_t size=20;
r = f1(a, size, f2, chg, &option);
```

Apresente, justificando de forma detalhada, o conteúdo da variável **r** e do *array* **a** após a execução do troço de código anterior, admitindo que o conteúdo daquele *array* antes da invocação da função `f1` é o indicado na alínea anterior.

Nota: a ausência de justificação do conteúdo do *array* **a** (após a execução do troço de código anterior) penalizará significativamente a avaliação desta alínea.

c) [1,5] No troço de programa seguinte consta ainda outro exemplo de utilização da função `f1`.

```
int *a, option=ALG, r;
(…) // Preenchimento do array dinâmico a.
size_t size=20;
r = f1(a, size, f3, f4, &option);
```

Pretende-se que a execução da função `f1` opere a seguinte transformação no *array* **a**: todos os elementos com números pares deverão passar a conter o valor da variável `option` .

Apresente, justificando, o código das funções `f3` e `f4` que realizam aquela transformação, tendo em conta que após a execução de `f1` a variável **r** deverá ter o valor 10.

Nota: desaconselha-se veementemente a escrita da definição destas funções sem a respetiva justificação porque penalizará significativamente a avaliação desta alínea.

3. [2 valores]

Escreva a função

```
int verifyWord(char *word, char *phrase);
```

que retorna 1 se a palavra identificada por `word` é idêntica a alguma das palavras existentes na *string* indicada por `phrase`; caso contrário, retorna 0. Na frase, considera-se palavra qualquer sequência de carateres que não sejam separadores.

Pode modificar o conteúdo da frase; por exemplo, pode isolar as palavras de modo a facilitar a comparação com a palavra procurada.

Os carateres separadores são `' '` (espaço), `'\t'` (*tab*), `'\n'` (*newline*), `'\r'` (*return*), `'\v'` (*vertical tab*) e `'\f'` (*form feed*), correspondendo à especificação da primitiva «`int isspace(int c);`» declarada em `ctype.h`.

4. [2 valores]

Considere um conjunto de módulos fonte, em linguagem C, e a utilização do comando "`gcc -c …`" para produzir os módulos compilados. A caixa ao lado contém os respetivos símbolos, apresentados pela ferramenta "`nm`". Considere também o módulo de aplicação "`prog.c`" cuja função `main` é reproduzida abaixo.

```
comp1.o:
0000000000000000 T compare1

comp2.o:
0000000000000000 T compare2

data.o:
                 U compare1
000000000000003e T dAdd
0000000000000029 T dCreate
000000000000013c T dDelete
00000000000000f9 T dPrint
00000000000000c1 T dSort
0000000000000000 t elemComp
                 U free
                 U iCreate
                 U iDelete
                 U malloc
                 U printf
                 U qsort
                 U realloc

info.o:
                 U free
0000000000000000 T iCreate
000000000000002b T iDelete
                 U malloc
                 U strdup
```

Admita que são usados no processo de compilação os *header files* `info.h`, `data.h`, `comp1.h`, e `comp2.h`, cada um contendo as assinaturas das funções públicas do respetivo módulo fonte (`.c`).

Além destes, existe ainda o *header file* `types.h` com a definição de tipos necessários aos diversos módulos, o qual é incluído em todos os *header files* anteriormente indicados.

a) [1] Escreva o *header file* `data.h`, assegurando o controlo de inclusão múltipla e supondo a assinatura das funções compatível com a utilização na função `main`.

b) [1] Tendo em conta as dependências existentes, escreva um *makefile* que produza, de forma eficiente, o executável com o nome "`prog`" a partir dos módulos fonte (`.c`) estritamente necessários.

`prog.c` (excerto)

```
int main(){
    Data *d = dCreate();
    char name[MAX_NAME];
    int num;
    while( scanf( "%d%*[ \t]%[^\n]", &num, name ) == 2 ){
        dAdd( d, name, num );
    }
    dSort( d );
    dPrint( d );
    dDelete( d );
}
```

5. [4,5 valores]

Pretende-se armazenar, em alojamento dinâmico, informação sobre os membros de uma organização. Os dados de cada pessoa são obtidos a partir de uma linha de texto iniciada pelo número de identificação, seguida de ponto-e-vírgula (‘`;`’), o nome, novamente ponto-e-vírgula e o cargo desempenhado.

Exemplos: “`123;Manuel Arruda;Presidente`”

“`234;Joaquim Salvaterra;Tesoureiro`”

“`246;Arlindo do Vale;Vogal`”

a) [1,5] Escreva a função

```
int lineParse(char *text, int *number, char *name, char *job);
```

destinada a separar e identificar os dados de uma linha, contidos na *string* indicada por `text`, e afetando as variáveis indicadas por `number`, `name` e `job`, respetivamente, com o número de identificação, o nome e o cargo. A função retorna: 1, em caso de sucesso; 0, se ocorrer insucesso, devido a conteúdo de `text` incompleto.

Assuma que o conteúdo da *string* `text` pode ser modificado e que os *arrays* passados em `name` e `job` têm dimensão suficiente para os conteúdos a depositar.

Sugere-se que use as funções `strtok` e `atoi` da biblioteca normalizada.

```
char *strtok(char *string, char *delimiters);
int atoi(char *string);
```

b) [1,5] Os dados são registados em elementos, alojados dinamicamente, com o tipo seguinte:

```
typedef struct{
  int number;          // Número de identificação
  char *name;          // Nome (string alojada dinamicamente)
  char *function;      // Cargo desempenhado (string alojada dinamicamente)
} Member;
```

Escreva a função

```
Member *memberCreate(char *text);
```

destinada a criar, em alojamento dinâmico, um elemento `Member` preenchido com os dados obtidos da *string* `text`. A função retorna o endereço do elemento criado ou NULL, em caso de insucesso. Neste caso, a função não deve deixar espaço inadequadamente alojado.

Deve utilizar a função `lineParse`. Sugere-se que use dois *arrays* auxiliares para armazenar temporariamente o nome e o cargo, dimensionados com a macro `MAX_LINE` que se assume já existir.

c) [1,5] Pretende-se armazenar o acesso aos dados num vetor (*array* dinâmico) de ponteiros para `Member`, com o descritor seguinte.

```
typedef struct{
  int count;       // Quantidade de ponteiros alojados e preenchidos no array
  Member **array;  // Array de ponteiros, alojado dinamicamente
} Staff;
```

Escreva a função

```
void staffAdd(Staff *staff, Member *member);
```

que adiciona ao vetor `staff` uma posição com o ponteiro para o elemento indicado por `member`.

Deve usar a função de biblioteca adequada para assegurar o espaço necessário ao novo conteúdo do *array* dinâmico.

6. [2,5 valores]

Admita que existe uma lista, simplesmente ligada, para referenciar os elementos do tipo `Member` definido no anteriormente, ordenada pelo número de identificação. Considere o nó de lista representado pelo tipo seguinte.

```
typedef struct listStaff{
  struct listStaff *next;   // ligação na lista
  Member *ref;              // referência: ponteiro para o elemento de dados
} ListStaff;
```

Supondo que se pretende dispor da lista por ordem oposta à inicialmente criada, escreva a função

```
void listInvert(ListStaff **headAddr);
```

que modifica as ligações de modo a inverter a posição relativa de todos os nós na lista.

7. [3 valores]

Admita que existe uma árvore binária de pesquisa para referenciar os elementos do tipo `Member` definido no anteriormente. A árvore é ordenada pelo campo `name` do elemento referenciado, alfabeticamente crescente de `left` para `right`. Considere o nó de árvore binária representado pelo tipo seguinte.

```
typedef struct bstStaff{
  struct bstStaff *left, *right; // ligações na árvore
  Member *ref;                   // referência: ponteiro para o elemento de dados
} BstStaff;
```

a) [2] Escreva a função

```
int bstGetNumber(BstStaff *root, char *name);
```

que procura, na árvore identificada por `root`, o nome indicado por `name` e retorna o respetivo número de identificação. Se o nome indicado não existir, deve retornar o valor «-1».

b) [1] Admita que existe uma árvore binária com 31 nós, perfeitamente balanceada, e é chamada a função `bstGetNumber`, podendo o nome existir ou não na árvore. Indique e justifique as quantidades mínima e máxima de nós da árvore que pode ser necessário observar, para comparar com a *string* do parâmetro `name`. Na identificação das quantidades pedidas, tenha em conta o comportamento do algoritmo implementado na alínea anterior (podendo este ser, por exemplo, recursivo ou iterativo).