Programação / Programação I - Exame da época normal

Licenciatura em Engenharia Informática e Curso Europeu de Informática

Duração: 2h30m 13/07/2012

Atenção: É obrigatório apresentar uma estratégia genérica para cada um dos exercícios.

---

1. Um ficheiro de texto contém resultados obtidos por diversas equipas. Cada linha do ficheiro armazena o resultado de um jogo no seguinte formato:

NomeEquipa1 – NomeEquipa2 : 3 – 2

Ao lado pode consultar um exemplo de um ficheiro de texto com este formato. Pode assumir que o nome das equipas é constituído apenas por uma palavra e que os caracteres separadores ('-', ':') surgem nas posições esperadas.

```
Equipa1 – Equipa2 : 1 – 0
Equipa2 – Equipa3 : 2 – 2
Equipa1 – Equipa4 : 4 – 0
Equipa2 – Equipa4 : 3 – 1
Equipa4 – Equipa3 : 4 – 0
Equipa5 – Equipa2 : 0 – 0
```

Desenvolva uma função em C que crie e preencha uma vector de estruturas com um resumo do desempenho das equipas. A informação de cada equipa deve ser armazenada numa estrutura do tipo `struct equipa`:

```
struct equipa{
      char nome[50];          // Nome da equipa
      int jogos;              // Jogos realizados
      int v, e, d;            // Vitórias, empates e derrotas
};
```

O vector de estruturas deve ser alocado dinamicamente. A função recebe o nome do ficheiro e o endereço de uma variável inteira como argumentos. Nesta variável inteira deve colocar o número de equipas que foram colocadas no vector dinâmico. Devolve como resultado o endereço do vector.

2. Considere a seguinte definição:

```
struct tree{
      int valor;
      struct tree *esq, *dir;
};
```

Escreva uma **função recursiva** em C

```
int conta_lim(struct tree *p, int min, int max);
```

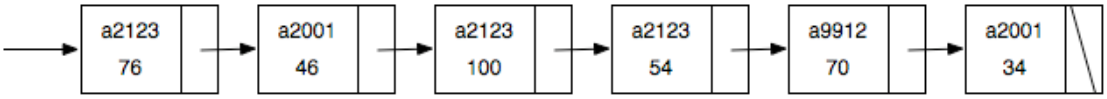
que conte quantos nós da **árvore binária ordenada** referenciada por *p* contêm valores que estejam dentro do intervalo [*min, max*]. A função devolve o número de nós contabilizado.

Atenção: a função sabe que a árvore está ordenada e deve tirar partido disso para tornar a pesquisa mais eficiente (i.e., não deve percorrer ramos onde sabe que não existem valores dentro do limite).

2. As notas que os alunos de uma turma obtiveram em diversos testes realizados ao longo do semestre estão armazenadas numa lista ligada não ordenada constituída por nós do tipo *teste*. O número de testes feito pelos diferentes alunos pode variar. No exemplo seguinte pode ver-se que o aluno *a2123* fez três testes, tendo obtido 76%, 100% e 54%.

```
typedef struct teste
     teste, *pteste;

struct teste {
      char id[15];
      int nota;
      pteste prox;
};
```



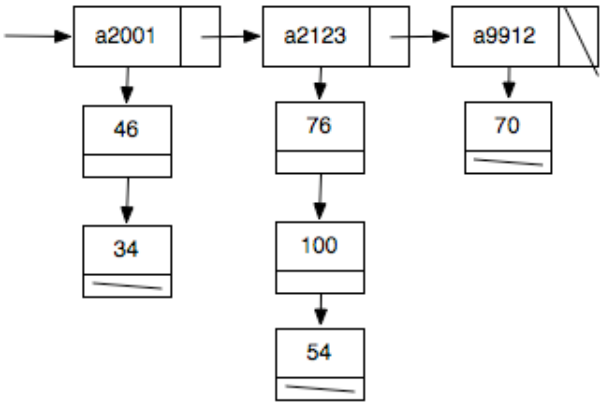
a) Desenvolva uma função em C que escreva no monitor a identificação do aluno que fez mais testes ao longo do ano. Se existirem vários alunos nestas circunstâncias, deve escrever a identificação do aluno que fez mais testes e teve melhor média. A função recebe um ponteiro para o início da lista como argumento.

b) Desenvolva uma função em C que transforme a lista ligada da alínea anterior numa lista de listas. Na nova estrutura dinâmica, a lista ligada principal é constituída por nós do tipo *aluno* (1 nó por aluno), **ordenados alfabeticamente** pelo campo *id*. A partir de cada um destes nós, acede-se a uma lista secundária constituída por elementos do tipo *av*. Em cada lista secundária ficam as provas feitas pelo respectivo aluno. A figura seguinte mostra como deve ficar organizada a estrutura dinâmica para o exemplo anterior:

```
typedef struct pessoa aluno, *paluno;
typedef struct aval av, *pav;

struct pessoa{
      char id[15];
      pav provas;
      paluno prox;
};

struct aval{
      int nota;
      pav prox;
};
```



A função recebe um ponteiro para o início da lista ligada simples como argumento e devolve um ponteiro para o início da lista de listas como resultado. Todo o espaço ocupado pela lista simples deve ser libertado.

Atenção: Ao resolver este problema deve chamar a seguinte função para o auxiliar na sua implementação (esta função já está implementada e só precisa de ser chamada):

```
int cria_insere_aluno(char *id, paluno* lista);
```

A função recebe o identificador de um aluno e a referência para o ponteiro da lista de listas (i.e., um ponteiro para um ponteiro). Verifica se o aluno com identificador *id* já existe na lista. Caso não exista, reserva o respectivo espaço e insere-o ordenadamente na lista. A função devolve 1 se tudo correu bem (o aluno já existia e não foi preciso fazer nada ou então o novo nó foi colocado com sucesso na lista), ou 0, se tiver existido um problema na alocação de memória.