Licenciatura em Engenharia de Electrónica e Telecomunicações e de Computadores

Programação Imperativa em C (e C++)

Licenciatura em Engenharia de Redes de Comunicação e Multimédia

Complementos de Programação

Teste global, 27 de Janeiro de 2010

1. [2] Considere o seguinte ficheiro fonte de um programa em C:

```
#define PRT(a,b) printf("%c=%d\n",*a,(b))
int main() { int d=31; PRT("%s",(d/3)); return 0; }
```

a) [1] Apresente o código resultante da pré-compilação deste ficheiro.

b) [1] Descreva os *warnings* apresentados na geração do programa e qual o resultado da sua execução.

2. [10] Para realizar um programa que contabiliza as ocorrências de valores inteiros positivos, considere os ficheiros `VC.h` e `VC.c` seguintes:

```
#include <stdio.h>

typedef struct _VC {
  unsigned value;   /*Valor*/
  unsigned counter; /*Contador de ocorrências*/
} VC;

void toString(VC *r, char *str);
void callAll(VC *v, int dim, void (*fx)(VC*));
void addValue(VC *v, int *dim, unsigned value);
void print(VC *r);
```

```
#include "VC.h"

void toString(VC *r, char *str)
{ sprintf(str,"%u|%u",r->value,r->counter); }

void callAll(VC *v, int dim, void (*fx)(VC*))
{ for( ; dim ; --dim,++v ) fx(v); }

void addValue(VC *v, int *dim, unsigned val) {
  int n = *dim;
  for( ; n ; ++v, --n )
    if (v->value==val) { v->counter++; return;}
  v->value = val;  v->counter = 1; ++*dim;
}
```

a) [2] Apresente uma implementação em IA-32 da função `callAll`.

Considere também o ficheiro `srepl.s` com a implementação em IA-32 da função `sreplace`:

```
sreplace:
  push ebp
  mov  ebp, esp
  mov  edx, [ebp+8]
  mov  cl,  [ebp+12]
  mov  ch,  [ebp+16]
  jmp  .L1
.Loop:
  inc  edx
.L1:
  mov  al, [edx]
  test al, al
  je   .End
  cmp  al, cl
  jne  .Loop
  mov  [edx],ch
  jmp  .Loop
.End:
  pop  ebp
  ret
```

b) [2] Implemente uma função em C equivalente à função `sreplace`.

c) [2] Faça um módulo `print.s` com a implementação em IA-32 da função `void print(VC *r)` que, usando apenas a função `puts` da biblioteca *standard* e as funções `toString` e `sreplace`, escreva no *standard output* os dois campos da estrutura apontada, mas separados por ':' em vez de '|'. Lembre-se que um valor inteiro a 32 bits não tem mais que 10 dígitos na base decimal.

d) [3] Usando as funções anteriores faça em C um módulo `prog.c` com a função `main` de um programa que, usando um *array* de estruturas `VC`, apresenta o número de ocorrências dos diferentes valores inteiros positivos lidos do *standard input*. A dimensão do *array* é indicada como argumento na linha de comando. O programa termina quando for lido um valor negativo.

e) [1] Faça o *makefile* que gera o programa da alínea anterior, compilando separadamente cada um dos módulos.

3. [8] Pretende-se que adicione à biblioteca de colecções `collection.a` o módulo `tree_iterator.o` que implementa um iterador para árvores binárias de pesquisa. De acordo com o `tree_iterator.h` apresentado a seguir, o iterador é constituído por um *array* de ponteiros (`a`), pela posição corrente do iterador (`cur`) e pela dimensão do *array* (`sz`). A implementação deste iterador não deverá ter preocupações de coerência entre os elementos referenciados pela árvore e os elementos referenciados pelo iterador.

```
#ifndef _tree_h
#define _tree_h

typedef struct bnode_t {
  struct bnode_t * left;
  struct bnode_t * right;
  void * pdata;
} BNode;

/*restantes funções…*/
#endif/*_tree_h*/
```

```
#ifndef _tree_iterator_h
#define _tree_iterator_h
#include "tree.h"
typedef struct tree_it_st {
        void **a;
        int cur;
        int sz;
} TreeIt;

TreeIt* init_tree_it(BNode *t);
void * next_tree_it(TreeIt *it);
void free_tree_it(TreeIt *it);
#endif/*_tree_iterator_h*/
```

A função `init_tree_it` cria um novo iterador alocando dinamicamente o espaço estritamente necessário para suportar todas as referências para os elementos da árvore `t`. O *array* terá as referências ordenadas de acordo com a ordem dos elementos na árvore (ordem crescente). O iterador deverá referenciar o primeiro elemento da árvore e deverá preservar a dimensão do *array*.

A função `next_tree_it` retorna o elemento de acordo com a posição corrente do `iterador it` e avança-o para o próximo elemento a retornar. Caso não tenha mais elementos retorna `NULL`.

A função `free_tree_it` liberta o espaço alocado dinamicamente no processo de criação do iterador `it`.

a) [1] Implemente a função `int getSize(BNode *t)`, local ao módulo `tree_iterator.c`, que retorna o número de elementos na árvore `t`.

b) [2] Implemente a função `void treetoa(BNode *t, void * a[], int *idx)` local ao módulo `tree_iterator.c`, que copia as referências dos elementos na árvore `t` para o *array* `a`, mantendo a ordem dos elementos.

c) [3] Implemente as funções `init_tree_it`, `next_tree_it` e `free_tree_it` de acordo com a especificação apresentada.

d) [1] Faça o *makefile* que cria a biblioteca estática `collection.a` sabendo que é formada pelos módulos `tree_iterator.o`, `binarytree.o` e `sortedlist.o`.

e) [1] Sabendo que a altura da árvore é o número de ligações a partir do ponteiro raiz até ao nó folha mais distante. Determine o número máximo de nós numa árvore perfeitamente balanceada com altura 6 e indique a altura correspondente ao pior caso de desbalanceamento para uma árvore com o mesmo número de nós.

Duração: 2 horas e 30 minutos

Bom teste!