MAC-122 – Princípios de Desenvolvimento de Algoritmos — Prova 1 – 20/9/2011

Identifique sua prova com nome e número USP. Numere as folhas.
Cada questão vale 2.5 pontos.

1. O tipo de primeira classe `string` é uma cadeia em memória dinâmica, com tamanho limitado só pela memória do computador. Qualquer caractere pode aparecer num desses strings, inclusive o `'\0'`.

   (a) Escreva um `typedef...string`, que permita declarar variáveis desse tipo. Uma variável nova dessas deve ser um string de comprimento 0. Uma possibilidade:

   ```
   typedef struct {
       int alloc; /* espaço alocado */
       int tam;   /* tamanho do string */
       char *c;   /* aponta uma área alocada com comprimento alloc */
   } string;
   ```

   Ao criar uma variável nova, todos os campos devem ser zerados.

   (b) Implemente a função

   `int string_cat(string *dest, string *orig)`

   copia o conteúdo de `*orig` para o fim de `*dest`. Ela deve retornar 0 se deu tudo certo e 1 se não houver memória suficiente para fazer essa cópia. Note que pode ser necessário obter mais espaço para `*dest`.

   ```
   int string_cat(string *dest, string *orig)
   {
       int comp;

       comp = orig->tam + dest->tam; /* tamanho do resultado */
       if (comp > dest->alloc) { /* falta espaço */
          realloc(dest->c, comp);
          dest->alloc = comp;
       }
       memcpy(dest->c + dest->tam, orig->c, orig->tam);
       dest->tam = comp;
   }
   ```

   Você pode usar à vontade funções da libc, sem se preocupar com os `#includes`; se não tiver certeza absoluta da sintaxe, coloque um comentário ao lado dizendo o que você acha que a função faz. Podem ser úteis:

```
void *malloc(size_t size);
void *realloc(void *ptr, size_t size);
void *memcpy(void *dest, const void *src, size_t n);
```

2. Em vários sistemas operacionais existe o conceito de *diretório* (também conhecido como *pasta*): uma lista (finita) de arquivos e diretórios. Vamos dizer que um arquivo ou diretório A está *embaixo* de um diretório B se A está na lista B ou A está embaixo de algum diretório que está embaixo de B.

   Aqui vai uma descrição simplificada das funções e estruturas usadas no Linux para acesso a diretórios:

   - Includes:
     ```
     #include <dirent.h>
     ```
     `dirent.h` define um tipo `DIR` — a definição exata não importa.
   - `DIR *opendir(const char *name);`

     `name` é o caminho de um diretório (`/usr/local/bin`, por exemplo). `opendir` devolve um apontador que deve ser usado com a função abaixo:
   - `struct dirent *readdir(DIR *dir);`

     onde (simplificando a realidade):
     ```
     struct dirent {
         unsigned char  d_type;      /* type of file */
         char           d_name[256]; /* filename */
     };
     ```
     Quanto ao valor `d_type`, está definida uma constante `DT_DIR`; se `d_type` tem esse valor, a dirent é um diretório, caso contrário é um "arquivo comum". Cada chamada de `readdir` com o mesmo parâmetro devolve um novo elemento do diretorio; quando a lista acaba, devolve NULL.

   Problema: escrever uma função que, dado o nome de um diretório, devolve o número de arquivos comuns embaixo dele.

   Você deve supor a seguinte simplificação da realidade: cada arquivo ou diretório aparece dentro de **um único** diretório.

A definição recursiva de *embaixo* pede uma solução recursiva:

o número de arquivos comuns embaixo de um diretório é o número de arquivos comuns nele mais o número de arquivos comuns embaixo de diretórios que estão nele. Um código natural é:

```
int conta_arquivos(char *nome)
{
    int num = 0;
    DIR *dir;
    struct dirent *d;

    dir = opendir(nome);
    while((d = readdir(dir)) != NULL)
       if(d->d_type == DT_DIR)
           num += conta_arquivos(d->d_name);
       else num++;
    return num;
}
```

Algo nessa linha vai ser aceito como correto. Só que esse código não funciona, por uma razão que pode não estar clara a partir das minhas explicações:

O conteúdo do campo `d_name` é *relativo* ao diretório `nome`. Assim, o nome real do diretório que deve ser aberto é a concatenação deles separados por uma barra: `nome/d_nome`. Por exemplo, se `nome = "/usr/include"` e `d->d_name = linux`, a chamada recursiva deveria corresponder a `conta_arquivos("/usr/include/linux")`, enquanto que no código acima corresponde a `conta_arquivos("linux")` e dá erro.

Aqui vai um código que funciona, com uns detalhes a mais.

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <dirent.h>
#include <string.h>
#include <error.h>
#include <errno.h>

int conta_arquivos(char *nome)
{
    int num = 0;
    DIR *dir;
    struct dirent *d;
    char *dname, *end;

    /*
     * reserva uma área para concatenar o nome dado com os dos subdiretórios
     * PATH_MAX é o tamanho máximo de um nome completo
     * no final deste bloco, dname contem nome seguido de /
     * e end aponta onde concatenar
     */
    dname = (char *) malloc(PATH_MAX);
    strcpy(dname, nome);
    end = dname + strlen(nome);
    *end++ = '/';

    dir = opendir(nome);

    /*
     * Existem várias razões para não se conseguir abrir um diretório.
     * Avisa o problema e segue em frente.
     */
    if(!dir) {
        error(0, errno, "não abriu %s",nome);
        return 0;
    }

    while(d = readdir(dir)) {
        /* Os subdiretórios . e .. sempre existem, mas não devem ser percorridos. */
        /* Fingir que não existem é parte da simplificação da realidade. */
        if(!strcmp(d->d_name, ".")) continue;
        if(!strcmp(d->d_name, "..")) continue;

        if(d->d_type == DT_DIR) {
            strcpy(end, d->d_name);
            num += conta_arquivos(dname);
        }
        else num++;
    }
    closedir(dir);
    free(dname);
    return num;
}

/* Driver para teste.  Chame o programa colocando um nome de diretório como parâmetro */
int main(int argc, char **argv)
{
    printf("%d\n", conta_arquivos(argv[1]));
}
```

3. Escreva uma função que recebe um apontador para uma lista ligada cujos itens são números inteiros, e devolve um apontador para uma lista do mesmo tipo, contendo os mesmos elementos que a lista original, mas com os números pares todos na frente dos números ímpares.

   Seu algoritmo deve ter complexidade linear em relação ao número de células da lista recebida; explique informalmente porque seu código tem essa propriedade.

   Não foi especificado se a lista tem ou não cabeça e cauda. Qualquer interpretação quanto a isso será aceita. Supondo uma lista simples, sem cabeça nem cauda:

```
typedef struct no {
   int Item;
   struct no *next;
} celula;

celula *parimpar(celula *lista)
{
        /* ideia: uma lista que cresce nas duas pontas */
        celula *inicio, fim;       /* para a lista nova */

        if (lista == NULL) return (celula *) NULL;
        /* é mais fácil iniciar com uma lista não vazia */
        inicio = fim = lista; /* pega a primeira célula */
        lista = lista->next;
        while( lista != NULL )
            if(lista->Item % 2) {
                /* caso ímpar */
                fim = fim->next = lista;
                lista = lista->next;
            }
            else {
                /* caso par */
                t = lista;
                lista = lista->next;
                t->next = inicio;
                inicio = t;
           }
        fim->next = NULL;
        return inicio;
}
```

   Outra ideia: criar duas listas e depois emendar.

4. Considere o seguinte código

```
int f(unsigned int n, unsigned int a[])
{
    int c = 0;
    while(--n)
      while(a[n]){
        c += a[n] % 2;
        a[n] = a[n] / 2;
    }
    return c;
}
```

(a) Explique o valor devolvido, qual sua relação com os parâmetros `a` e `n`. Suponha `n > 0`.

Os valores de `a` que são olhados vão de `n-1` a `1`. Para cada entrada, os bits `1` da sua representação binária são contados em `c`. Assim, o valor devolvido é

> o número total de bits 1 nas representações binárias de $\mathtt{a}[1], \mathtt{a}[2], \ldots, \mathtt{a}[\mathtt{n}-1]$.

(b) Estime, em função dos dados, quantas vezes a atribuição à variável `c` dentro dos `while`'s é executada.

Cada elemento de `a` é dividido por 2 enquanto não der 0; o número de vezes que isso ocorre para um inteiro k é $\lceil \lg(k+1) \rceil$. Assim, o número pedido é

$$T = \sum_{i=1}^{n-1} \lceil \lg(\mathtt{a}[\mathtt{i}] + 1) \rceil.$$

Estimativa mais grosseiras, mas mais manejáveis (e aceitas como resposta):

$$\begin{aligned} T &= O\left(\sum_{i=1}^{n-1} \lg(\mathtt{a}[\mathtt{i}] + 1)\right) \\ &= O((n-1)\lg M) \qquad (\text{onde} \quad M = \max_i \mathtt{a}[\mathtt{i}] + 1) \\ &= O(n \lg M). \end{aligned}$$