Lista de Exercícios para P1 – INF1007 (Prog2) – 15/04/2013

ATENÇÃO: O material a seguir não sofreu revisão e pode conter erros.

EXERCÍCIO 1:

Considere as informações dos candidatos inscritos em um concurso. Para não haver favorecimento na hora da correção, cada candidato recebe uma identidade secreta, que é o que deve constar na prova desse candidato.

A identidade secreta é uma cadeia gerada com a seguinte lei de formação: Os 8 primeiros caracteres correspondem à data de nascimento (ano, mês, dia), os caracteres seguintes correspondem às iniciais do nome do candidato e após um *, vem o primeiro nome da mãe do candidato.

Exemplo: para o candidato chamado Rui Abreu Soares, nascido em 27/05/1982, filho de Clara, teríamos a seguinte cadeia: "19820527RAS*CLARA". Assuma que todas as letras são maiúsculas, e que não há acentos, cedilhas, etc.

1.a) Escreva em C a função **exibeNascimento**, que recebe a identidade secreta de um candidato, e mostra na tela a data de nascimento desse candidato, no formato dia/mês/ano.

1.b) Escreva em C a função **obtemMae**, que recebe a identidade secreta e retorna, sem alocar dinamicamente, nem usar qualquer outro vetor de char, um ponteiro para a cadeia contendo o nome da mãe do candidato.

1.c) Escreva em C a função **geraIdentidade** para gerar a identidade secreta de um candidato a a partir dos dados lidos do teclado. Inicialmente a função lê do teclado o nome completo do candidato (que deve se digitado todo em maiúsculas e apenas com um branco entre cada uma das palavras que constituem um nome, tendo o nome completo tamanho máximo de 80 caracteres). Em seguida deve ser solicitada a data de nascimento do candidato, a ser digitada com o formato "dia/mês/ano", e, por fim, será lido o nome da mãe do candidato. A função deve retornar uma nova cadeia, alocada dinamicamente com o tamanho exato necessário, com a identidade secreta do candidato. Caso não seja possível criar a nova cadeia , a função deve retornar NULL.

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

void exibeNascimento (char *ident);
char *obtemMae(char *ident);
char *geraIdentidade (void);

int main (void)
{
	char *id;
	id = geraIdentidade();
	printf("\n%s",id);
	exibeNascimento(id);
	printf("\n%s",obtemMae(id));
	free (id);
	getch();
	return 0;
}


void exibeNascimento (char *ident)
{	char dtnasc[11];
	int i;

	dtnasc[0]=ident[6];
	dtnasc[1]=ident[7];
	dtnasc[2]='/';
	dtnasc[3]=ident[4];
	dtnasc[4]=ident[5];
	dtnasc[5] = '/';
	for (i=6; i<10; i++)
		dtnasc[i]=ident[i-6];
	dtnasc[10]= '\0';
	printf("\n%s", dtnasc);
}

char *obtemMae(char *ident)
{	/* assumindo que ident é uma identidade válida */
	char *sufixo;
	int tam, i;
	tam = strlen (ident);
	i=tam - 1;
	while ( ident[i]!= '*')
		i--;
	sufixo= ident + i +1;  /* ou s = &ident[i+1] */
	return sufixo;
}
```

```c
char *geraIdentidade (void)
{
	char nome[81];
	char iniciais[10];
	char nasc[11];
	char mae[81];
	int tam, i, j;
	char *novasenha;

	printf("Nome completo:");
	scanf(" %[^\n]", nome);

	printf("\nNascimento (dd/mm/aaaa):");
	scanf(" %[^\n]", nasc);
	printf("\nPrimeiro nome da mae:");
	scanf(" %[^\n]", mae);
	iniciais[0]=nome[0];
	j = 1;
	for (i=1; nome[i]!= '\0' ; i++)
		if (nome[i]==' ')
		{	iniciais[j]= nome[i+1];
			j++;
		}
	iniciais[j]='\0';
	nasc[2]=nasc[5]='\0';

	tam = 8 + strlen(iniciais) + 1+ strlen(mae) +1;
	novasenha = (char *)malloc (tam * sizeof(char));
	if (novasenha == NULL)
		return NULL;
	strcpy(novasenha, &nasc[6]);
	strcat(novasenha, &nasc[3]);
	strcat (novasenha, nasc);
	strcat (novasenha,iniciais);
	strcat(novasenha, "*");
	strcat (novasenha, mae);

	return novasenha;

}

/* 19820527RAS*CLARA  */
```

EXERCÍCIO 2:

A avaliação de um vendedor de uma loja é mapeada no tipo estruturado Avaliacao abaixo:

```
struct avaliacao
{       float   valorVendidoNoMes;
        int     numeroDeFaltas;
};
typedef  struct avaliacao  Avaliacao;
```

Os dados de um vendedor são representados pelo tipo estruturado Vendedor abaixo:

```
struct vendedor
{       int             inscricao;
        char            nome[51];
        float           salário-base;
        Avaliacao       aval;
};
typedef struct  vendedor Vendedor;
```

2.a) Escreva em C a função **calculaExtra** que recebe uma avaliação e retorna um valor (float) correspondente a um valor extra a ser acrescido ao salário do vendedor no pagamento do mês. Esse valor extra deve ser computado da seguinte forma:

- Se o número de faltas é menor do que 2 e o valor vendido no mês é maior do que 2000, a função retorna 180.
- Se o número de faltas é maior ou igual a do 2 e menor do que 4 e o valor vendido é maior do que 2000, a função retorna 150.
- Se o número de faltas é maior ou igual a 4 e menor do que 6, mas o valor vendido é maior do que 5000, a função retorna 120.
- Do contrário a função retorna 0.

2.b) Para a folha de pagamento da loja é utilizado um vetor de vendedores . Escreva em C a função **calculaValorTotalDaFolha** que recebe o número de vendedores e o vetor dos vendedores da loja e calcula qual o valor total que a loja deverá disponibilizar para pagamentos naquele mês, que será o resultado da soma dos valores a serem pagos a todos os vendedores (valor para cada vendedor : salario + extra). A função deve utilizar a função calculaExtra do item anterior.

2.c) Escreva a versão recursiva da função anterior.

2.d) Escreva em C a função **obtemRelacaoDemitidos**, que recebe o número de vendedores e o vetor dos vendedores da loja e retorna um novo vetor de inteiros, alocado com o tamanho exato necessário, com as inscrições dos vendedores com número de faltas >= 10, que serão demitidos. A função deve também devolver em uma variável inteira, cujo endereço foi fornecido com argumento na chamada da função, o número de vendedores que serão demitidos. Caso não seja possível criar o novo vetor ou caso não existam vendedores com mais de 10 faltas, a função deve retornar NULL.

2.e) Assumindo, para facilitar, que os valores vendidos no mês são todos diferentes, escreva a função **obtemMelhorVendedor**, que recebe o número de vendedores e o vetor dos vendedores da loja e retorna, sem alocar dinamicamente, nem usar qualquer outro vetor de char, um ponteiro para a cadeia contendo o nome do melhor vendedor da loja (o que tem o maior valor vendido no mês).

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

struct avaliacao
{      float  valorVendidoNoMes;
       int    numeroDeFaltas;
};
typedef  struct avaliacao  Avaliacao;

struct vendedor
{      int          inscricao;
       char         nome[51];
       float        salarioBase;
       Avaliacao    aval;
};
typedef struct  vendedor Vendedor;


float calculaExtra(Avaliacao av);
float calculaValorTotalDaFolha (Vendedor *vvend, int numVend);
void exibeUmVendedor(Vendedor vd);
int *obtemRelacaoDemitidos (Vendedor *vvend, int numVend, int *pnumDem);
char *obtemMelhorVendedor (Vendedor *vvend, int numVend);

int main (void)
{
       Vendedor vend[5] = { {111, "ana", 1500, {3000,3}},
                            {222, "bia", 1000, {4000,5}},
                            {333, "leo", 1500, {1000,12}},
                            {444, "edu", 1000, {3000,1}},
                            {555, "cae", 1500, {2000,13}}};



       int i;
       float totalDaFolha;
       char *nomeDoMelhor;
       int *vdemitido;
       int numDemitidos;
```

```
	for (i=0; i<5; i++)
		exibeUmVendedor(vend[i]);
	totalDaFolha = calculaValorTotalDaFolha(vend, 5);

	nomeDoMelhor= obtemMelhorVendedor (vend,5);
	printf("Melhor vendedor: %s\n", nomeDoMelhor);

	vdemitido = obtemRelacaoDemitidos (vend, 5, &numDemitidos);
	for (i=0; i<numDemitidos; i++)
		printf("%d\n", vdemitido[i]);
	getch();
	return 0;
}


float calculaExtra(Avaliacao av)
{
	if ( av.numeroDeFaltas < 2 && av.valorVendidoNoMes >2000)
		return 180;
	if ( av.numeroDeFaltas >=2 && av.numeroDeFaltas < 4 &&
	     av.valorVendidoNoMes >2000)
		return 150;
	if ( av.numeroDeFaltas >=4 && av.numeroDeFaltas < 6 &&
	     av.valorVendidoNoMes >5000)
		return 120;
	return 0;

}

float calculaValorTotalDaFolha (Vendedor *vvend, int numVend)
{	float tot = 0;
	int i;
	for (i=0; i<numVend; i++)
		tot= tot + vvend[i].salarioBase + calculaExtra(vvend[i].aval);
	printf("Total da folha : %.2f\n", tot);
	return tot;
}

void exibeUmVendedor(Vendedor vd)
{	printf("%d - %s - sal base: %.2f - valVendido:%.2f - numFaltas:%d\n",
	     vd.inscricao, vd.nome, vd.salarioBase, vd.aval.valorVendidoNoMes,
vd.aval.numeroDeFaltas);
}
```

```
int *obtemRelacaoDemitidos (Vendedor *vvend, int numVend, int *pnumDem)
{
	int *vdemit;
	int  cont = 0;
	int i,j;
	for (i=0; i<numVend; i++)
		if (vvend[i].aval.numeroDeFaltas >= 10)
			cont ++;
	*pnumDem = cont;
	if (cont == 0)
		return NULL;
	vdemit = (int *)malloc (cont * sizeof(int));
	if (vdemit == NULL)
		return NULL;
	j=0;
	for (i=0; i<numVend; i++)
		if (vvend[i].aval.numeroDeFaltas >= 10)
		{	vdemit[j] = vvend[i].inscricao;
			j++;
		}
	return vdemit;
}

char *obtemMelhorVendedor (Vendedor *vvend, int numVend)
{	char *melhor;
	int indMelhor;
	int i;
	indMelhor = 0;
	for (i=0;i<numVend; i++)
		if (vvend[i].aval.valorVendidoNoMes >
		     vvend[indMelhor].aval.valorVendidoNoMes)
			indMelhor = i;
	melhor = vvend[indMelhor].nome;
	return melhor;
}
```

EXERCÍCIO 3:

Considere o cadastro com os códigos dos produtos comercializados por uma empresa, armazenados em um vetor bidimensional de char (cada código é uma cadeia, em que os 4 últimos dígitos representam o ano de lançamento do produto).

3.a) Escreva a função **contaMaisAntigos**, que recebe o número de produtos e o vetor (bidimensional) dos códigos dos produtos, recebe também um ano (como uma cadeia de 4 dígitos) e retorna o número de códigos com ano menor do que o ano recebido.

3.b) Escreva a versão recursiva da função do item anterior.

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

#define NUMMAXCOD 10
#define TAMMAXCOD 21

int contaMaisAntigos(char cod[][TAMMAXCOD], int n, char *ano);

int main (void)
{
	char codigo[NUMMAXCOD][TAMMAXCOD]= {
			"ESTMADMET1988", "CAMAFERRO2003",
			"CADMADRODAS2005", "MESATAMPOVD1999", "BARMAD1995",
			"SOFATEC3LUG1992", "BAURODAS2001",
			 "CARRINHOJARDIM2004","MESAINOX1997", "TRIPRATMAD2001"};
	int totMaisAnt;

	totMaisAnt = contaMaisAntigos(codigo, 10, "2000");
	printf("Ha %d codigos anteriores a 2000", totMaisAnt);
	getch();
}

int contaMaisAntigos(char cod[][TAMMAXCOD], int n, char *ano)
{	int i , tot, tam;

	char *final;
	tot=0;
	for (i=0; i<n; i++)
	{	tam = strlen(cod[i]);
		final = &(cod[i][tam-4]);
		printf("%s\n",final);
		if(strcmp(final, ano) < 0)
			tot++;
	}
	return tot;
}
```

EXERCÍCIO 4:

4) O cadastro dos funcionários de uma certa empresa de transportes (a TRAN) utiliza um vetor de ponteiros para o tipo estruturado Funcionario abaixo que representa um funcionário dessa empresa.

struct funcionario

{ char motorista[51];

int qtDeMultas; /* quantidade de multas desse motorista */

};

typedef struct funcionario Funcionario;

4.a) Escreva a função criaFuncionario, que recebe o nome e o a quantidade DE MULTAS DE um novo funcionário e cria um novo funcionário com esses dados, utilizando alocação dinâmica. Caso não seja possível criar tal funcionário, a função retorna NULL.

4.b) Escreva em C a função **contaPioresMotoristas**, que recebe o número de funcionários da empresa, o vetor de de ponteiros para funcionários, e retorna o número de motoristas com quantidade de multas >10.

4.c) Escreva a versão recursiva da função do item anterior (4.a) .

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


Funcionario * criaFuncionario( char *nm, int *mt);
int contaPioresMotoristas(int num, Funcionario **vf);
int contaPioresMotoristasRec(int num, Funcionario **vf);

Funcionario * criaFuncionario( char *nm, int *mt)
{      Funcionario * novo;
       novo = (Funcionario *) malloc (sizeof (Funcionario));
       if (novo == NULL)
              return NULL;
       strcpy(novo->nome, nm);
       novo -> qtDeMultas = mt;
       return novo;
}


int contaPioresMotoristas(int num, Funcionario **vf)
{      int i, tot;
       tot = 0;
       for (i=0; i<num; i++)
              if (vf[i]->qtDeMultas >10)
                     tot++;

       return tot;
}
```

```c
/* VERSÃO RECURSIVA */

int contaPioresMotoristasRec(int num, Funcionario **vf)
{
	if (num == 0)
		return 0;

	if (vf[0]->qtDeMultas >10)
		return ( 1 + contaPioresMotoristasRec( num-1, &vf[1]));
	else
		return (contaPioresMotoristasRec( num-1, &vf[1]);

}
```

EXERCÍCIO 5: (Único sem solução)

5) Considere o tipo estruturado Endereco abaixo:

```
struct endereço
{       char    rua[31];
        int     numero;
        char    complemento[15];
        char    bairro[22];
};
typedef struct endereço Endereco;
```

5.a) Escreva a função **leDadosDeEndereco** que recebe uma (ponteiro para) estrutura Endereço e a preenche com os dados lidos do teclado como uma cadeia, que deve ser digitada no formato:

minha_rua numero meu_complemento meu_bairro.

Exemplo: rua_marques_de_sao_vicente 345 apto_501_bloco_2 jardim_botanico.

(Considere que existe um único branco entre cada componente do endereço)

Para os próximos itens da questão 5, considere, para representar um imóvel do cadastro de uma imobiliária, o tipo estruturado Imovel a seguir:

```
struct imovel
{       char            tipo[15]; /* casa, apto, casa de vila, outro */
        Endereco        ender;
        float           valor;
        char            proprietário;
};
typedef struct imovel Imovel;
```

5.b) Escreva em C a função **exibeNoBairro**, que recebe o número de imóveis da imobiliária, o vetor de imóveis e o nome de um bairro, e exibe os dados de todos os imóveis nesse bairro.

5.c) Escreva a versão recursiva da função **exibeNoBairro**.

5.d) Escreva a função **contaNoBairro** que recebe o número de imóveis da imobiliária, o vetor de imóveis e o nome de um bairro, e retorna o número de imóveis nesse bairro.

5.e) Escreva a versão recursiva da função **contaNoBairro**

5.f) Escreva a função **exibeEcontaNoBairro**, que recebe o número de imóveis da imobiliária, o vetor de imóveis e o nome de um bairro, e e exibe os dados de todos os imóveis nesse bairro, retornando o número de imóveis nesse bairro.

5.g) Escreva a versão recursiva da função **exibeEcontaNoBairro**.