Universidade Federal de Ouro Preto – UFOP
Instituto de Ciências Exatas e Biológicas – ICEB
Departamento de Computação – DECOM
Disciplina: Algoritmos e Estruturas de Dados I – CIC102
Professor: David Menotti (menottid@gmail.com)

Lista de Exercícios sobre
Tipos Abstratos de Dados (TAD) e Estruturas de Dados em C

GABARITO
Contribuição: Antonio Carlos Nazaré Júnior

1) Escreva uma especificação de tipos abstratos de dados (TAD) para os números complexos, $a + bi$, onde $abs(a + bi)$ é $sqrt(a^2 + b^2)$, $(a + bi) + (c + di)$ é $(a + c) + (b + d)i$, $(a + b) * (c + di)$ é $(a * c – b * d) + (a * d + b * c)i$ e $–(a + bi)$ é $(-a) + (-b)i$. Então, implemente (cm C/C++ ou Java) números complexos, conforme especificado acima, usando estruturas com partes reais e complexas. Escreva rotinas para somar, multiplicar e negar (inverter) tais números.

```
#include <stdio.h>
#include <iostream>
using namespace std;

struct complexo
{
       double real;
       double imaginario;
};

complexo soma(complexo* num1, complexo* num2)
{
         complexo tempsoma;
         tempsoma.real = num1->real + num2->real;
         tempsoma.imaginario = num1->imaginario + num2->imaginario;
         return tempsoma;
}

complexo multiplica(complexo* num1, complexo* num2)
{
         complexo tempmult;
         tempmult.real = num1->real * num2->real - num1->imaginario * num2->imaginario;
         tempmult.imaginario = num1->real * num2->imaginario + num1->imaginario * num2->real;
         return tempmult;
}

complexo inverte(complexo* pnum)
{
         complexo temp;
         temp.real = (-1) * pnum -> real;
         temp.imaginario = (-1) * pnum -> imaginario;
         return temp;
}

void escreve(complexo* num)
{
         printf("%.2f + %.2fi\n", num -> real, num -> imaginario);
}

void setcomplex(complexo* pnum, double real, double imagin)
{
     pnum -> real = real;
     pnum -> imaginario = imagin;
}


int main()
{

    complexo comp1;
```

```
    complexo comp2;
    double r1,r2;


    printf ("Digite a parte real do número Complexo 1: ");
    cin >> r1;
    printf ("Digite a parte imaginaria do número Complexo 1: ");
    cin >> r2;
    setcomplex(&comp1, r1, r2);

    printf ("Digite a parte real do número Complexo 2: ");
    cin >> r1;
    printf ("Digite a parte imaginaria do número Complexo 2: ");
    cin >> r2;

    setcomplex(&comp2, r1, r2);

    printf ("Número Complexo 1: ");
    escreve(&comp1);
    printf ("Número Complexo 2: ");
    escreve(&comp2);

    printf ("Soma de Complexo 1 e Complexo 2: ");
    escreve(&soma(&comp1,&comp2));

    printf ("Multiplicação de Complexo 1 e Complexo 2: ");
    escreve(&multiplica(&comp1,&comp2));

    printf ("Inversão do Complexo 1: ");
    escreve(&inverte(&comp1));

    system("PAUSE");
}
```

2) Vamos supor que um número real seja representado por uma estrutura em C, como esta:

```
struct realtype
{
      int left;
      int right;
};
```

onde *left* e *right* representam os dígitos posicionados à esquerda e à direita do ponto decimal, respectivamnete. Se *left* for um inteiro negativo, o número real representado será negativo.

a. Escreva uma rotina para inserir um número real e criar uma estrutura representado esse número;
b. Escreva uma função que aceite essa estrutura e retorne o número real representado por ela.
c. Escreva rotinas add, substract e multiply que aceitem duas dessas estruturas e definam o valor de uma terceira estrutura para representar o número que seja a soma, a diferença e o produto, respectivamente, dos dois registros de entrada.

```
#include <math.h>
#include <iostream>
using namespace std;

struct realtype
{
```

```
int left;
int right;
};

realtype criaReal(double num)
{
    realtype tempReal;
    tempReal.left = (int) num;
    tempReal.right = abs((int)((num-tempReal.left)*10000));
    return tempReal;
}

double retornaDoub(realtype* real)
{
       double temp = 0;
       temp += real->left + (real->right/10000.0);
       return temp;
}

realtype add(realtype* real1, realtype* real2)
{
         realtype tempReal;
         tempReal.left = (real1->left+real2->left)+((real1->right+real2->right)/10000);
         tempReal.right = ((real1->right+real2->right)%10000);
         return tempReal;

}

realtype multiply(realtype* real1, realtype* real2)
{
         realtype tempReal;
         long tempSoma=0;
         tempSoma += real1->right*(real2->left*10000);
         tempSoma += real1->right*(real2->right);
         tempSoma += (real1->left*10000)*(real2->left*10000);
         tempSoma += (real1->left*10000)*(real2->right);
         tempReal.left = tempSoma/100000000;
         tempReal.right = tempSoma%100000000;
         return tempReal;
}

void escreve(realtype* real)
{
     printf("%d.%d\n", real->left, real->right);
}

int main()
{
    double num;
    realtype real1;
    realtype real2;
    printf("Digite um número Real 1: ");
    cin >> num;
    real1 = criaReal(num);
    printf("Digite um número Real 2: ");
    cin >> num;
    real2 = criaReal(num);
    escreve(&real1);
    escreve(&real2);
    printf("Soma: ");
    escreve(&add(&real1, &real2));
    printf("Multiplicação: ");
    escreve(&multiply(&real1, &real2));
    system("PAUSE");
}
```

3) Suponha que um inteiro precise de quatro bytes, um número real precise de oito bytes e um caractere precise de um byte. Pressuponha as seguintes definições e declarações:

```
struct nametype
{
      char first[10];
      char midinit;
      char last[20];
};

struct person
{
      struct nametype name;
      int             birthday[2];
      struct nametype parentes[2];
      int             income;
      int             numchildren;
      char            address[20];
      char            city[10];
      char            state[2];
};

struct person p[100];
```

se o endereço inicial de p for 100, quais serão os endereços inicias (em bytes) de cada um dos seguintes?

a. `p[10]` 1410
b. `p[200].name.midinit` 200
c. `p[20].income` 80
d. `p[20].address[5]` 2000
e. `p[5].parents[1].last[10]` 1000

4) Suponha dois vetores, um de registros de estudantes e outro de registros de funcionários. Cada registro de estudante contém membros para um último nome, um primeiro nome e um índice de pontos de graduação. Cada registro de funcionário contém membros para um último nome, um primeiro nome e um salário. Ambos os vetores são classificados em ordem alfabética pelo último e pelo primeiro nome. Dois registros com o último e o primeiro nome iguais não aparecem no mesmo vetor. Escreva uma função em C para conceder uma aumento de 10% a todo funcionário que tenha um registro de estudante cujo índice de pontos de graduação seja maior que 3.0.

```
#include <stdio.h>
#include <iostream>
#include <string>
using namespace std;

#define tam 5

struct estudante
{
      string ultNome;
      string priNome;
      double pontos;
```

```
};

struct funcionario
{
       string ultNome;
       string priNome;
       double salario;
};

void aumento(estudante est[], funcionario func[], int quant)
{
     for (int i=0; i<quant; i++)
         if (est[i].pontos > 3.0 )
            func[i].salario += func[i].salario*0.1;
}

void ordena_estudante(estudante est[])
{
     estudante estTemp;
     for (int i=0; i<tam-1; i++)
     {
         int posMenor = i;
         for (int j=i+1; j<tam; j++)
         {
             if (est[j].ultNome <= est[posMenor].ultNome)
                if ((est[j].ultNome == est[posMenor].ultNome) && (est[j].priNome <
est[posMenor].priNome))
                    posMenor = j;
             else
                 posMenor=j;
         }
         if (posMenor != i)
         {
            estTemp = est[i];
            est[i] = est[posMenor];
            est[posMenor] = estTemp;
         }
     }
}

void ordena_funcionario(funcionario func[])
{
     funcionario funcTemp;
     for (int i=0; i<tam-1; i++)
     {
         int posMenor = i;
         for (int j=i+1; j<tam; j++)
         {
             if (func[j].ultNome <= func[posMenor].ultNome)
                if ((func[j].ultNome == func[posMenor].ultNome) && (func[j].priNome <
func[posMenor].priNome))
                    posMenor = j;
             else
                 posMenor=j;
         }
         if (posMenor != i)
         {
            funcTemp = func[i];
            func[i] = func[posMenor];
            func[posMenor] = funcTemp;
         }
     }
}



int main()
{
    estudante est[tam];
    funcionario func[tam];
    printf("Entre com informações de %d estudantes\n\n", tam);
```

```
    for (int i=0; i<tam; i++)
    {
        printf("\n---------------\n");
        printf("Estudante %d \n", i+1);
        printf("---------------\n");
        cout << "Ultimo Nome: ";
        getline(cin,est[i].ultNome);
        cout << "Primeiro Nome: ";
        getline(cin,est[i].priNome);
        if (est[i].priNome==est[i].ultNome)
        {
            printf("PROIBIDO NOMES IGUAIS\n");
            i--;

        }
        else
        {
            printf("Pontos: ");
            cin >> est[i].pontos;
            cin.get();
        }
    }
    printf("\n\nEntre com informações de %d funcionarios\n\n", tam);
    for (int i=0; i<tam; i++)
    {
        printf("\n---------------\n");
        printf("Funcionario %d \n", i+1);
        printf("---------------\n");
        cout << "Ultimo Nome: ";
        getline(cin,func[i].ultNome);
        cout << "Primeiro Nome: ";
        getline(cin,func[i].priNome);
        if (func[i].priNome==func[i].ultNome)
        {
            printf("PROIBIDO NOMES IGUAIS\n");
            i--;

        }
        else
        {
            printf("Salário: ");
            cin >> func[i].salario;
            cin.get();
        }
    }
    ordena_estudante(est);
    ordena_funcionario(func);
    cout << "\nESTUDANTES DEPOIS DE CLASSIFICADOS\n";
    for (int i=0; i<tam; i++)
    {
        cout << est[i].ultNome << " - " << est[i].priNome << endl;
    }

    cout << "\nFUNCIONARIOS DEPOIS DE CLASSIFICADOS\n";
    for (int i=0; i<tam; i++)
    {
        cout << func[i].ultNome << " - " << func[i].priNome << endl;
    }
    aumento(est,func,tam);
    system("PAUSE");
}
```

5) Escreva uma função semelhante à do exercício anterior, mas pressupondo que os registros dos funcionários e estudantes sejam mantidos em dois arquivos externos classificados, em vez de em dois vetores classificados.

```
#define _CRT_SECURE_NO_WARNINGS // evitar warnings

#include <stdio.h>
#include <iostream>
#include <string>
using namespace std;

#define tam 5
#define MAXNOM 20

struct estudante
{
	char ultNome[MAXNOM];
	char priNome[MAXNOM];
	double pontos;
};

struct funcionario
{
	char ultNome[MAXNOM];
	char priNome[MAXNOM];
	double salario;
};

void aumento(FILE* pE, FILE* pF, int quant)
{
	struct estudante   est;
	struct funcionario fun;

	fseek( pE, 0L, SEEK_SET);
	fseek( pF, 0L, SEEK_SET);

	for (int i=0; i<quant; i++)
	{
		fread(&fun,sizeof(struct estudante),1,pF);

		fseek( pE, 0L, SEEK_SET);
		for(int j=0; j<quant; j++)
		{
			fread(&est,sizeof(struct estudante),1,pE);

			if ( (strcmp(fun.priNome,est.priNome) == 0) &&
(strcmp(fun.ultNome,est.ultNome) == 0) )
			{
				if (est.pontos > 3.0 )
				{
					fun.salario += fun.salario*0.1;
					fseek( pF,sizeof(struct funcionario)*i, SEEK_SET);
					fwrite(&fun,sizeof(struct funcionario),1,pF);
				}
			}
		}
	}
}

void ordena_estudante(FILE* pE)
{
	fseek( pE, 0L, SEEK_SET);

	estudante est1,est2,estMenor;
	for (int i=0; i<tam-1; i++)
	{
		int posMenor = i;
		fseek(pE,sizeof(struct estudante)*i,SEEK_SET);
		fread(&est1,sizeof(struct estudante),1,pE);

		estMenor = est1;
		for (int j=i+1; j<tam; j++)
		{
			fread(&est2,sizeof(struct estudante),1,pE);
```

```
                        if (strcmp(est2.ultNome,estMenor.ultNome) <= 0)
                        {
                                if ( !((strcmp(est2.ultNome,estMenor.ultNome)==0) &&
(strcmp(est2.priNome,estMenor.priNome)>0)) )
                                {
                                        estMenor = est2;
                                        posMenor = j;
                                }
                        }
                }

                fseek(pE,sizeof(struct estudante)*i,SEEK_SET);
                fwrite(&estMenor,sizeof(struct estudante),1,pE);

                fseek(pE,sizeof(struct estudante)*posMenor,SEEK_SET);
                fwrite(&est1,sizeof(struct estudante),1,pE);
        }
}

void ordena_funcionario(FILE* pF)
{
        fseek( pF, 0L, SEEK_SET);

        funcionario fun1,fun2,funMenor;
        for (int i=0; i<tam-1; i++)
        {
                int posMenor = i;
                fseek(pF,sizeof(struct funcionario)*i,SEEK_SET);
                fread(&fun1,sizeof(struct funcionario),1,pF);

                funMenor = fun1;
                for (int j=i+1; j<tam; j++)
                {
                        fread(&fun2,sizeof(struct funcionario),1,pF);
                        if (strcmp(fun2.ultNome,funMenor.ultNome) <= 0)
                        {
                                if ( !((strcmp(fun2.ultNome,funMenor.ultNome)==0) &&
(strcmp(fun2.priNome,funMenor.priNome)>0)) )
                                {
                                        funMenor = fun2;
                                        posMenor = j;
                                }
                        }
                }

                fseek(pF,sizeof(struct funcionario)*i,SEEK_SET);
                fwrite(&funMenor,sizeof(struct funcionario),1,pF);

                fseek(pF,sizeof(struct funcionario)*posMenor,SEEK_SET);
                fwrite(&fun1,sizeof(struct funcionario),1,pF);
        }
}

int main()
{
        FILE *pE, *pF;
        struct estudante   est;
        struct funcionario fun;

        // criando arquivos
        pE = fopen("estudante.dat","wb+");
        if (pE == NULL) { fprintf(stderr,"erro ao tentar criar arquivo estudante.dat\n");
exit(0); }

        pF = fopen("funcionario.dat","wb+");
        if (pE == NULL) { fprintf(stderr,"erro ao tentar criar arquivo funcionario.dat\n");
exit(0); }

        printf("Entre com informações de %d estudantes\n\n", tam);
        for (int i=0; i<tam; i++)
        {
```

```
                printf("\n---------------\n");
                printf("Estudante %d \n", i+1);
                printf("---------------\n");
                cout << "Ultimo Nome: ";
//              scanf("%s",est.ultNome);
                cout << "Primeiro Nome: ";
//              scanf("%s",est.priNome);
                if (i==0) {strcpy(est.priNome,"Andre"          );strcpy(est.ultNome,"GOMES
ASSENCO"   );est.pontos=0.7;}
                if (i==1) {strcpy(est.priNome,"Angelo"         );strcpy(est.ultNome,"FERREIRA
ASSIS"  );est.pontos=0.8;}
                if (i==2) {strcpy(est.priNome,"Antonio Carlos" );strcpy(est.ultNome,"DE NAZARE
JUNIOR");est.pontos=1.0;}
                if (i==3) {strcpy(est.priNome,"Carlos Henrique");strcpy(est.ultNome,"PACHECO DE
SOUZA");est.pontos=0.5;}
                if (i==4) {strcpy(est.priNome,"Daniel"         );strcpy(est.ultNome,"ROCHA
GUALBERTO" );est.pontos=0.9;}

        if (strcmp(est.priNome,est.ultNome) == 0)
        {
            printf("PROIBIDO NOMES IGUAIS\n");
            i--;
        }
        else
        {
            printf("Pontos: ");
//                  scanf("%f",&est.pontos);
                    fwrite(&est,sizeof(struct estudante),1,pE);
        }
    }

    printf("\n\nEntre com informações de %d funcionarios\n\n", tam);
    for (int i=0; i<tam; i++)
    {
        printf("\n---------------\n");
        printf("Funcionario %d \n", i+1);
        printf("---------------\n");
        cout << "Ultimo Nome: ";
//        scanf("%s",fun.ultNome);
        cout << "Primeiro Nome: ";
//        scanf("%s",fun.priNome);
            if (i==0) {strcpy(fun.priNome,"Alex"           );strcpy(fun.ultNome,"AMORIM
DUTRA"       );fun.salario=0.2;}
            if (i==1) {strcpy(fun.priNome,"Alisson"        );strcpy(fun.ultNome,"OLIVEIRA
DOS SANTOS");fun.salario=0.8;}
            if (i==2) {strcpy(fun.priNome,"Antonio Augusto");strcpy(fun.ultNome,"ALVES
PEREIRA"      );fun.salario=1.0;}
            if (i==3) {strcpy(fun.priNome,"Brayan"         );strcpy(fun.ultNome,"VILELA
ALVES"       );fun.salario=0.5;}
            if (i==4) {strcpy(fun.priNome,"Bruno"          );strcpy(fun.ultNome,"CERQUEIRA
HOTT"     );fun.salario=0.9;}

            if (strcmp(fun.priNome,fun.ultNome)== 0)
            {
                    printf("PROIBIDO NOMES IGUAIS\n");
                    i--;
            }
            else
            {
                    printf("Salário: ");
//                  scanf("%f",&fun.salario);
                    fwrite(&fun,sizeof(struct funcionario),1,pF);
            }
    }

    ordena_estudante(pE);
    ordena_funcionario(pF);

    fseek( pE, 0L, SEEK_SET);
    cout << "\nESTUDANTES DEPOIS DE CLASSIFICADOS\n";
    for (int i=0; i<tam; i++)
```

```
        {
                fread(&est,sizeof(struct estudante),1,pE);
                cout << est.ultNome << " - " << est.priNome << endl;
        }

        fseek( pF, 0L, SEEK_SET);
        cout << "\nFUNCIONARIOS DEPOIS DE CLASSIFICADOS\n";
        for (int i=0; i<tam; i++)
        {
                fread(&fun,sizeof(struct funcionario),1,pF);
                cout << fun.ultNome << " - " << fun.priNome << endl;
        }

        aumento(pE,pF,tam);

        fclose(pE);
        fclose(pF);

        system("PAUSE");
}
```

6) Usando a representação de números racionais apresentada abaixo, escreva rotinas para somar, subtrair e dividir tais números.

```
typedef struct
{
      int numerator;
      int denominator;
} RATIONAL;
```

```
struct RATIONAL
{
      int numerator;
      int denominator;
};
```

```
#include <stdio.h>
#include <iostream>

struct RATIONAL
{
int numerator;
int denominator;
};

void setRational(RATIONAL* r, int num, int den)
{
     r->numerator = num;
     r->denominator = den;
}


int mmc(int d1, int d2)
{
    int maior;
    int menor;
    int mmctemp;
    if (d1==d2)
       return d1;
    else if (d1>d2)
         {
            maior = d1;
            menor = d2;
         }
        else
         {
            maior = d2;
            menor = d1;
         }
    mmctemp = maior;
    while (mmctemp%menor != 0)
          mmctemp += maior;
```

```
    return mmctemp;
}

int MDC(int num1, int num2){
  int tempresto;

  while(num2 != 0){
    tempresto = num1 % num2;
    num1 = num2;
    num2 = tempresto;
  }

  return num1;
}

void reduzRat(RATIONAL* num1)
{
     int maximo = MDC(num1->numerator, num1->denominator);
     num1->numerator = num1->numerator / maximo;
     num1->denominator = num1->denominator / maximo;
}

RATIONAL soma(RATIONAL* num1, RATIONAL* num2)
{
    RATIONAL temp_rat;
    int minimo = mmc(num1->denominator, num2->denominator);
    temp_rat.denominator = minimo;
    temp_rat.numerator = (minimo/num1->denominator*num1->numerator);
    temp_rat.numerator += (minimo/num2->denominator*num2->numerator);
    reduzRat(&temp_rat);
    return temp_rat;
}

RATIONAL sub(RATIONAL* num1, RATIONAL* num2)
{
    RATIONAL temp_rat;
    int minimo = mmc(num1->denominator, num2->denominator);
    temp_rat.denominator = minimo;
    temp_rat.numerator = (minimo/num1->denominator*num1->numerator);
    temp_rat.numerator -= (minimo/num2->denominator*num2->numerator);
    reduzRat(&temp_rat);
    return temp_rat;
}

RATIONAL div(RATIONAL* num1, RATIONAL* num2)
{
    RATIONAL temp_rat;
    temp_rat.denominator = num1->denominator*num2->numerator;
    temp_rat.numerator = num2->denominator*num1->numerator;
    reduzRat(&temp_rat);
    return temp_rat;
}


int equal(RATIONAL* num1, RATIONAL* num2)
{
    reduzRat(num1);
    reduzRat(num2);
    if (num1->numerator==num2->numerator)
       if(num1->denominator==num2->denominator)
          return 0;
       else
          return 1;
    else
        return 1;
}

int equal2(RATIONAL* num1, RATIONAL* num2)
{

    if (num1->numerator*num2->denominator==num2->numerator*num1->denominator)
```

```
        return 0;
    else
        return 1;
}

void escreveRational(RATIONAL r)
{
     printf("%d/%d\n", r.numerator, r.denominator);
}


int main()
{
    RATIONAL rat1, rat2, rat3;
    int num1, num2;
    printf("Digite o numerador do 1° Racional: ");
    scanf("%d", &num1);
    printf("Digite o denominador do 1° Racional: ");
    scanf("%d", &num2);
    setRational(&rat1, num1, num2);
    printf("Digite o numerador do 2° Racional: ");
    scanf("%d", &num1);
    printf("Digite o denominador do 2° Racional: ");
    scanf("%d", &num2);
    setRational(&rat2, num1, num2);
    printf("Os racionais digitados foram: ");
    escreveRational(rat1);
    escreveRational(rat2);
    printf("Soma entre eles: ");
    escreveRational(soma(&rat1, &rat2));
    printf("Subtração entre eles: ");
    escreveRational(sub(&rat1, &rat2));
    printf("Divisão entre eles: ");
    escreveRational(div(&rat1, &rat2));
    if (equal(&rat1, &rat2)==0)
       printf("Eles são iguais\n");
    if (equal2(&rat1, &rat2)!=0)
       printf("Eles são diferentes\n");
    system("PAUSE");
    /*
    Levando em conta o como argumento da Função complexidade o número de
    comparações temos a tabela:
    metodo | pior caso | caso medio | melhor caso
    ---------------------------------------------
    equal  |     2     |    1,5     |      1
    equal2 |     1     |     1      |      1
    ---------------------------------------------
    Por isso o metodo equal2 é preferivel.
    */

}
```

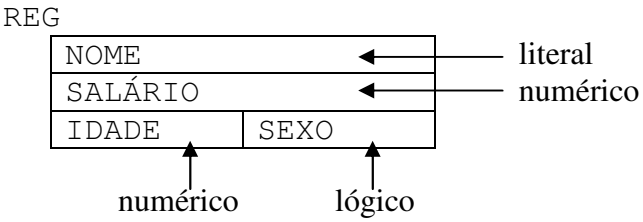
7) Implemente uma função *equal* que determine se dois números racionais, *r*1 e *r*2, são iguais ou não reduzindo primeiramente *r*1 e *r*2 a seus termos mínimos e verificando em seguida a igualdade.

Veja solução da questão 6

8) Um método alternativo para implementar a função *equal* citada na questão anterior seria multiplicar o denominador de cada número pelo numerador do outro e testar a igualdade dos dois produtos. Escreva uma função *equal2* para implementar esse algoritmo. Qual dos dois métodos é preferível?

Veja solução da questão 6

9) Definir e declarar o registro cuja representação gráfica é dada a seguir.

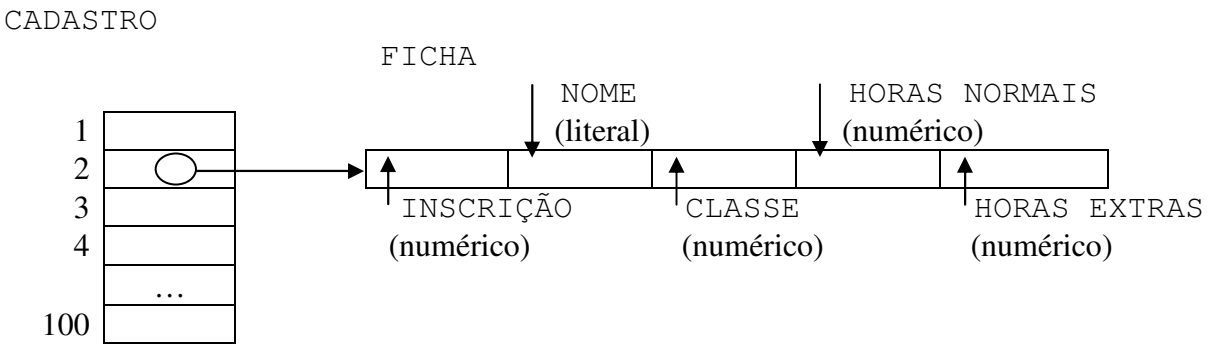



```
#include <stdio.h>
struct REG
{
        char nome[100];
        double sal;
        int idade;
        bool sexo;
};

int setSal(REG* r)
{
    printf("Digite o salário: ");
    scanf("%f", &r->sal);
}

int main()
{
    REG registro;
    setSal(&registro);
}
```

10) Escrever uma função para atribuir um valor ao campo de nome SALÁRIO do registro REG, descrito no exercício anterior. Escreva também um programa (função *main*) para utilizar a função criada.

Veja solução da questão 9

11) Uma indústria faz a folha mensal de pagamentos de seus empregados baseada no seguinte: Existe uma tabela com os dados do funcionário



Fazer um algoritmo que processe a tabela e emita, para cada funcionário seu contracheque cujo formato é dado a seguir:

Universidade Federal de Ouro Preto – UFOP
Instituto de Ciências Exatas e Biológicas – ICEB
Departamento de Computação – DECOM
Disciplina: Algoritmos e Estruturas de Dados I – CIC102
Professor: David Menotti (menottid@gmail.com)

```
NÚMERO DE INSCRIÇÃO:      NOME:
SALÁRIO HORAS NORMAIS:
SALÁRIO HORAS EXTRAS:
DEDUÇÃO INSS:
  SALÁRIO LÍQUIDO:
```

O salário de referência deverá ser lido previamente.

O salário referente às horas extras é calculado acrescentando 30% ao salário-hora normal.

O desconto do INSS é de 11% do salário bruto (salário correspondente às horas normais trabalhadas + salário correspondente às horas extras).

Para o cálculo do salário, considerar que existem duas classes de funcionários, a classe 1, cujo salário é 1,3 vezes o salário de referência, e a classe 2, cujo salário é 1,9 vezes o salário de referência.

```c
#include <stdio.h>
#define tam 5
struct FICHA
{
       int insc;
       char nome[50];
       int classe;
       int horasNormais;
       int horasExtras;
};

void le_ficha(FICHA* fich)
{
     printf("Número de Inscrição: ");
     scanf("%d", &fich->insc);
     printf("Nome: ");
     scanf("%s", &fich->nome);
     printf("Classe: ");
     scanf("%d", &fich->classe);
     printf("Horas Normais: ");
     scanf("%d", &fich->horasNormais);
     printf("Horas Extras: ");
     scanf("%d", &fich->horasExtras);
}

void emite_contracheque(FICHA* fich, double salRef)
{
     double sal;
     if (fich->classe==1)
        sal = salRef*1,3;
     else
        sal = salRef*1,9;
     double salNormal = sal*fich->horasNormais;
     double salExtra = sal*fich->horasExtras;
     double INSS = (salNormal+salExtra) - 0.11*(salNormal+salExtra);
     printf("\n=======================================================\n");
     printf("Número de Inscrição %5d Nome: %50s\n", fich->insc, fich->nome);
     printf("Salário Horas Normais: %f\n", salNormal);
     printf("Salário Horas Extras: %f\n", salExtra);
     printf("Dedução do INSS: %f\n", INSS);
     printf("Salário Líquido: %f\n", (salNormal+salExtra)-INSS);
     printf("=======================================================\n");
}

int main()
{
    FICHA cadastro[tam];
    double salRef;
```

```
    for (int i=0; i<tam; i++)
        le_ficha(&cadastro[i]);
    printf("Digite o valor do salário de referencia: ");
    scanf("%f", &salRef);
    for (int i=0; i<tam; i++)
        emite_contracheque(&cadastro[i], salRef);
}
```

12) Para evitar erros de digitação de sequências de números de importância fundamental, como a matrícula de um aluno, o CPF do Imposto de Renda, o número de conta bancária, geralmente adiciona-se ao número um dígito verificador. Por exemplo, o número de matrícula 811057 é usado como 8110573, onde 3 é o dígito verificador, calculado da seguinte maneira:
    a. Cada algarismo do número e multiplicado por um peso começando de 2 e crescendo de 1 em 1, da direita para a esquerda:
    $$8 \times 7, 1 \times 6, 1 \times 5, 0 \times 4, 5 \times 3, 7 \times 2;$$
    b. Somam-se as parcelas obtidas:
    $$56 + 6 + 5 + 0 + 15 + 14 = 96;$$
    c. Calcula-se o resto da divisão desta soma por 11:
    96 divido por 11 dá resto 8 ($96 = 8 \times 11 + 8$);
    d. Subtrai-se de 11 o resto obtido:
    $$11 - 8 = 3;$$
    e. Se o valor encontrado for 10 ou 11, o dígito verificador será 0; nos outros casos, o dígito verificar é o próprio resto da divisão.

    Escrever um algoritmo capaz de:
    1) Ler um conjunto de registros contendo, cada um, o número de uma conta bancária, o dígito verificador deste número, o saldo da conta e o nome do cliente. O último registro, que não deve ser considerado contém o número de conta igual a zero.
    2) Utilizando o esquema de verificação acima, imprimir duas listas de clientes distintas no seguinte formato de saída:

```
CONTAS DE NÚMERO CORRETO
413599-7    987,30  Jacinto Pereira
111118-     121,99  Campos Sales
06
...

CONTAS DE NÚMERO ERRADO
765432-1    335,66  Júnia Faria
...
```

13) Defina um Tipo Abstrato de Dados `TMatriz`, para representar matrizes quadradas de tamanho n. Implemente as operações para somar e multiplicar 2 matrizes. Implemente ainda a operação do cálculo da matriz inversa.

14) Você deverá implementar um tipo abstrato de dados `TConjunto` para representar conjuntos de números inteiros. Seu tipo abstrato deverá armazenar os elementos do conjunto e o seu tamanho *n*. Considere que o tamanho máximo de um conjunto é 20 elementos e use arranjos de 1 dimensão (vetores) para a sua implementação. Seu TAD deve possuir procedimentos (ou funções quando for o caso) para:
    a. criar um conjunto vazio;
    b. ler os dados de um conjunto;
    c. fazer a união de dois conjuntos;
    d. fazer a interseção de dois conjuntos;
    e. verificar se dois conjunto são iguais (possuem os mesmos elementos);
    f. imprimir um conjunto;

Exercícios extraídos de (Referências)

[1] Aaron M. Tenenbaum, Yedidyah Langsam, Moshe J. Augenstein, *Estruturas de Dados Usando C*, Makron Books/Pearson Education, 1995.
[2] Aaron M. Tenenbaum, Yedidyah Langsam, Moshe J. Augenstein, *Data Strcutures Using C*, Prentice-Hall International Editions, 1995.
[3] Harry Farrer, Christiano Gonçalves Becker, Eduardo Chaves Faria, Helton Fábio de Matos, Marcos Augusto dos Santos, Miriam Lourenço Maia, *Algoritmos Estruturados*, LTC Editora, 3ª. edição, Rio de Janeiro, 1999.