Universidade Federal de Goiás – UFG
Instituto de Informática – INF
Bacharelados (Núcleo Básico Comum)

Algoritmos e Estruturas de Dados 1 – 2022/1

Lista de Exercícios nº 03 – Tipo Abstrato de Dados (TAD)

Sumário

1 Conjuntos de Números Naturais **4**

2 Manipulando Datas **7**

Observações:

- A resolução de cada um dos exercícios desta lista pressupõe a utilização do conceito de *Tipo Abstrato de Dados* (TAD) durante a implementação utilizando a linguagem de programação C;
- Tendo em vista que, por característica de construção do ambiente *Sharif Judge System* do INF/UFG utilizado nesta disciplina, **apenas um único arquivo** pode ser enviado como proposta de solução para um problema, tal arquivo deverá ter a estrutura apresentada na figura a seguir;
- Detalhando: o único arquivo com a extensão `.c` deverá conter o que *deveria estar* em dois arquivos: o `tad.h` e o próprio `tad.c`;
- O uso do arquivo `tad.h` significa que a função `main()` elaborada como proposta de solução para o problema deve somente utilizar operações presentes neste arquivo.

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

<TADs: conteúdo do(s) arquivo(s) .h>

<TADs: conteúdo do(s) arquivo(s) .c , sem #include "TAD.h">

int main () {
        <seu código>
}
```

Veja o exemplo a seguir:

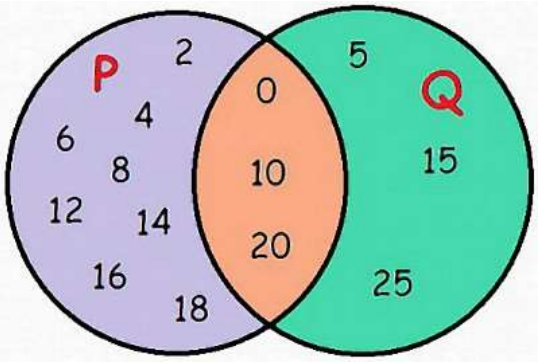
```
//=========================================================================
// Arquivo ponto.h
//=========================================================================
//
#include <stdio.h>
 #include <stdlib.h>
 #include <math.h>
#include <stdbool.h>

 typedef struct ponto Ponto;
 Ponto*  ponto_cria(float x, float y, bool visibilidade);
void   ponto_libera(Ponto* p);
void   ponto_acessa(Ponto* p, float* x, float* y);
void   ponto_atribui(Ponto* p, float x, float y);
float  ponto_distancia(Ponto* p1, Ponto* p2);
 void   ponto_oculta(Ponto* p);
 void   ponto_mostra(Ponto* p);
void   ponto_move(Ponto* p, float x, float y);
//
//=========================================================================
// Arquivo ponto.c
//=========================================================================
//
struct ponto {
   float x;
   float y;
   bool visibilidade;
};
//
// Cria um ponto
//
 Ponto* ponto_cria (float x, float y, bool visibilidade) {
    Ponto* p = (Ponto*) malloc(sizeof(Ponto));
    if (p != NULL) {
       p->x = x;
       p->y = y;
       p- >visibilidade = visibilidade;
    }

    return (p);
}
//
// Libera (desaloca) um ponto...
//
 void ponto_libera (Ponto* p) {
    if (p != NULL) {
       free(p);
    }
}
//
// Acessa um ponto, coletando suas coordenadas
//
 void ponto_acessa (Ponto* p, float* x, float* y) {
    if (p != NULL) {
      *x=p->x;
      *y=p->y;
    }
}
//
// Atribui coordenadas a um ponto, modificando –o
//
 void ponto_atribui (Ponto* p, float x, float y) {
    if (p != NULL) {
       p->x=x;
       p->y=y;
    }
}
//
// Retorna a distancia entre dois pontos
//
 float ponto_distancia (Ponto* p1, Ponto* p2) {
    float dx = p1->x - p2->x;
    float dy = p1->y - p2->y;
    return (sqrt(dx*dx+dy*dy));
}
```

```
76
77  //
78  // Oculta (torna invisivel) o ponto
79  //
80  void ponto_oculta (Ponto* p) {
81
82     p->visibilidade = false;
83  }
84  //
85  // Mostra (torna visivel) o ponto
86  //
87  void ponto_mostra (Ponto* p) {
88
89     p->visibilidade = true;
90  }
91
92  void ponto_move(Ponto * p, float x, float y) {
93     //
94     // Codigo para movimentacao do ponto
95     //
96  }
97  //
98  // Corpo principal
99  //
100
101 int main(){
102    float xp,yp,xq,yq,d;
103    Ponto *p,*q;
104
105    printf("digite as coordenadas x e y para o ponto 1: ");
106    scanf("%f %f",&xp,&yp);
107    printf("digite as coordenadas x e y para o ponto 2: ");
108    scanf("%f %f",&xq,&yq);
109    p  = ponto_cria(xp,yp, true);
110    q  = ponto_cria(xq,yq, true);
111    d = ponto_distancia(p,q);
112    ponto_acessa(p,&xp,&yp); ponto_acessa(q,&xq,&yq);
113    printf("Distancia entre os pontos (%.2f,%.2f) e (%.2f,%.2f) = %.5f\n",xp,yp,xq,yq,d);
114    ponto_libera(p); ponto_libera(q);
115    return (0);
116 }
```

./programas/pontoCompleto.c



1 Conjuntos de Números Naturais

(++) A linguagem C não possui um *tipo de dado* que seja capaz de representar a ideia de *conjunto finito* conforme a acepção Matemática do termo, ou seja, "*Uma coleção finita de elementos (entes ou componentes), na qual a ordem e a repetição destes elementos é irrelevante e, por isso, desconsiderada.*".
Escreva, em C, um programa que seja capaz de representar um *conjunto de números naturais* por meio do uso do conceito de Tipo Abstrato de Dado (TAD).
O programa deve implentar, no mínimo, as seguintes operações fundamentais:

1. criar um conjunto *C*, inicialmente *vazio*:
   `int  criaConjunto(C);`
   retornando `SUCESSO` ou `FALHA`.
   A falha ocorre se não for possível, por alguma ocorrência, criar o conjunto *C*.

2. verificar se o conjunto *C* é vazio:
   `int conjuntoVazio(C);`
   retornando `TRUE` ou `FALSE`.

3. incluir o elemento *x* no conjunto *C*:
   `int insereElementoConjunto(x, C);`
   retornando `SUCESSO` ou `FALHA`.
   A falha acontece quando o elemento *x* já está presente no conjunto *C* ou, por algum outro motivo, a inserção não pode ser realizada.

4. excluir o elemento *x* do conjunto *C*:
   `int excluirElementoConjunto(x, C);`
   retornando `SUCESSO` ou `FALHA`.
   A falha acontece quando o elemento *x* não está presente no conjunto *C* ou, por algum outro motivo, a remoção não pode ser realizada.

5. calcular a cardinalidade do conjunto *C*:
   `int tamanhoConjunto(C);`
   retornando a quantidade de elementos em *C*. O valor 0 (zero) indica que o conjunto está vazio.

6. determinar a quantidade de elementos do conjunto *C* que são maiores que *x*:
   `int maior(x, C);`
   O valor 0 (zero) indica que todos os elementos de *C* são maiores que *x*.

7. determinar a quantidade de elementos do conjunto *C* que são menores que *x*:
   `int menor(x, C);`
   O valor 0 (zero) indica que todos os elementos de *C* são menores que *x*.

8. verificar se o elemento *x* pertence ao conjunto *C*:
   `int pertenceConjunto(x, C);`
   retornando `TRUE` ou `FALSE`.

9. comparar se dois conjuntos, C_1 e C_2 são idênticos:
   `int conjuntosIdenticos(C1, C2);`
   retornando `TRUE` ou `FALSE`.

10. identificar se o conjunto C_1 é subconjunto do conjunto C_2:
    `int subconjunto(C1, C2);`
    retornando `TRUE` ou `FALSE`.

11. gerar o complemento do conjunto C_1 em relação ao conjunto C_2:
    `Conjunto complemento(C1, C2);`
    retornando um *conjunto* que contém os elementos de C_2 que não pertencem a C_1.
    Se todos os elementos de C_2 estão em C_1, então deve retornar um conjunto vazio.

12. gerar a união do conjunto C_1 com o conjunto C_2:
    `Conjunto uniao(C1, C2);`
    retornando um *conjunto* que contém elementos que estão em C_1 ou em C_2.

13. gerar a intersecção do conjunto C_1 com o conjunto C_2:
    `Conjunto interseccao(C1, C2);`
    retornando um *conjunto* que contém elementos que estão em C_1 e, simultaneamente, em C_2.
    Se não houver elementos comuns deverá retornar um conjunto vazio.

14. gerar a diferença entre o conjunto C_1 e o conjunto C_2:
    `Conjunto diferenca(C1, C2);`
    retornando um *conjunto* que contém elementos de C_1 que não pertencem a C_2.
    Se todos os elementos de C_1 estão em C_2 deve retornar um conjunto vazio.

15. gerar o conjunto das partes do conjunto *C*:
    `Conjunto conjuntoPartes(C);`

16. mostrar os elementos presentes no conjunto *C*:
    `void mostraConjunto(C, ordem);`
    Mostrar, no dispositivo de saída, os elementos de *C*.
    Se `ordem` for igual a CRESCENTE, os elementos de *C* devem ser mostrados em ordem crescente. Se ordem for igual a DECRESCENTE, os elementos de *C* devem ser mostrados em ordem decrescente.
    Observação: Como o dispositivo típico de saída é o monitor de vídeo, o(a) programador(a) tem liberdade para definir como os elementos serão dispostos nele. Por exemplo: dez ou vinte elementos por linha. Noutro exemplo: o programa definirá quantos elementos mostrar, por linha, de acordo com o número de elementos existentes no conjunto a ser apresentado.

17. copiar o conjunto C_1 para o conjunto C_2:
    `int copiarConjunto(C1, C2);`
    retornando `SUCESSO` ou `FALHA`.
    A falha acontece quando, por algum motivo, não é possível copiar os elementos do conjunto C_1 para o conjunto C_2.

18. destruir o conjunto *C*:
    `int destroiConjunto(C);`

retornando `SUCESSO` ou `FALHA`.
A falha acontece quando, por algum motivo, não é possível eliminar o conjunto *C* da memória.

Observações: Considere que:

- `SUCESSO` = 1; `FALHA` = 0;
- `TRUE` = 1; `FALSE` = 0;
- `CRESCENTE` = 1; `DECRESCENTE` = 0;
- qualquer conjunto poderá ter no máximo 1.000.000 (um milhão) de elementos, ou seja, esta é a *cardinalidade máxima* de um conjunto. Se qualquer operação resultar num conjunto com cardinalidade maior, então a função correspondente deverá retornar um *conjunto vazio* (se ela retorna um conjunto) ou `FALHA` (se ela retorna `SUCESSO` ou `FALHA`);
- a biblioteca `limits.h` da linguagem C contém duas constantes para denotar quais são o *menor* e o *maior* `long int` que pode ser utilizado no ambiente computacional em que o programa está sendo elaborado. São elas: `LONG_MIN` e `LONG_MAX`. Elas deverão ser, respectivamente, o menor e o maior número que podem ser armazenados num conjunto qualquer do programa;
- os nomes das funções anteriormente apresentados no texto devem ser obedecidos, ou seja, o código-fonte C elaborado deverá obrigatoriamente utilizá-los. É claro que outras funções acessórias podem ser criadas livremente pelo(a) programador(a).

Entradas e Saídas

Não serão fornecidas entradas/saídas para testes, pois o(a) estudante deverá apenas submeter o código-fonte por ele(a) elaborado no *Sharif Judge System* do INF/UFG.
O programa elaborado deverá ter um *menu* que permita ao usuário selecionar cada uma das operações supramencionadas, executá-la e, em seguida, retornar ao *menu* para escolher uma nova opção.
Para *finalizar o programa* o usuário deverá fornecer um entrada especial. Por exemplo, o número 0 (zero) como opção no *menu*.
O(A) estudante terá liberdade para escolher como implementar a funcionalidade de *menu*.

2 Manipulando Datas

 (++)

É inquestionário que a capacidade de *manipular datas* é de extrema importância em muitas aplicações práticas na área de processamento de dados. Infelizmente nem sempre há, numa determinada linguagem de programação que se está utilizando para o desenvolvimento de aplicações, uma *biblioteca* com variadas funções para realizar a manipulação de datas.

Considere que você está participando do desenvolvimento de uma *biblioteca* para esta finalidade, sendo que ela deverá ser integralmente escrita em C e conter pelo menos as seguintes funções, expressas por seus cabeçalhos: `data.h`.

1. `Data * criaData (unsigned int dia, unsigned int mes, unsigned int ano);`
   Cria, de maneira dinâmica, uma *data* a partir dos valores para dia, mês e ano fornecidos.

2. `Data * copiaData (Data d);`
   Cria uma *cópia* da data `d`, retornando-a.

3. `void liberaData (Data * d);`
   Destrói a data indicada por `d`.

4. `Data * somaDiasData (Data d, unsigned int dias);`
   Retorna uma data que é `dias` dias posteriores à data `d`.
   Por exemplo, fornecendo a data `d = 16/03/2020` e `dias = 5`, retornará a data `21/03/2020`.

5. `Data * subtrairDiasData (Data d, unsigned int dias);`
   Retorna uma data que é `dias` dias anteriores à data `d`.
   Por exemplo, fornecendo a data `d = 16/03/2020` e `dias = 15`, retornará a data `01/04/2020`.

6. `void atribuirData (Data * d, unsigned int dia, unsigned int mes, unsigned int ano);`
   Atribui, à data `d`, a data `dia/mes/ano` especificada.
   Se não for possível, então faz com que `d` seja alterada para `NULL`.

7. `unsigned int obtemDiaData (Data d);`
   Retorna a componente `dia` da data `d`.

8. `unsigned int obtemMesData (Data d);`
   Retorna a componente `mes` da data `d`.

9. `unsigned int obtemAnoData (Data d);`
   Retorna a componente `ano` da data `d`.

10. `unsigned int bissextoData (Data d);`
    Retorna `TRUE` se a data pertence a um ano bissexto. Do contrário, retorna `FALSE`.

11. `int comparaData (Data d1, Data d2);`
    Retorna MENOR se `d1` < `d2`, retorna IGUAL se `d1` = `d2` ou retorna MAIOR, se `d1` > `d2`.

12. `unsigned int numeroDiasDatas (Data d1, Data d2);`
    Retorna o número de dias que existe entre as datas `d1` e `d2`.
    Se `d1` = `d2`, então o número de dias é igual a 0 (zero). Do contrário, será um número estritamente positivo.

13. `unsigned int numeroMesesDatas (Data d1, Data d2);`
    Se `d1` e `d2` estão no mesmo mês/ano, então o número de meses é igual a 0 (zero). Do contrário, será um número estritamente positivo.

14. `unsigned int numeroAnosDatas (Data d1, Data d2);`
    Se `d1` e `d2` estão no mesmo ano, então o número de anos é igual a 0 (zero). Do contrário, será um número estritamente positivo.

15. `unsigned int obtemDiaSemanaData (Data d);`
    Retorna o *dia da semana* correspondente à data `d`.
    Considerando que `DOMINGO` = 1; `SEGUNDA-FEIRA` = 2; ... ; `SÁBADO` = 7.

16. `char * imprimeData (Data d, char * formato);`
    Retorna uma *string* com a data "*formatada*" de acordo com o especificado em `formato`.
    Se `formato` = "ddmmaaaa", então a *string* retornada deverá apresentar os dois dígitos do dia, os dois dígitos do mês e os quatro dígitos do ano, nesta ordem, e separados por uma (/ – barra). Por exemplo: "12/11/2019".
    Se `formato` = "aaaammdd", então a *string* retornada deverá apresentar os quatro dígitos do ano, os dois dígitos do mês e os dois dígitos do dia, nesta ordem, e separados por uma (/ – barra).
    Por exemplo: "2019/11/12".
    De maneira análoga, são válidas as seguintes *strings* de formatação:

    - "aaaa";
    - "mm";
    - "dd";
    - "ddmm".

Entrada e Saídas

Não serão fornecidas entradas/saídas para testes, pois o(a) estudante deverá apenas submeter o código-fonte por ele(a) elaborado no *Sharif Judge System* do INF/UFG.
O programa elaborado deverá ter um *menu* que permita ao usuário selecionar cada uma das operações supramencionadas, executá-la e, em seguida, retornar ao *menu* para escolher uma nova opção.
Para *finalizar o programa* o usuário deverá fornecer um entrada especial. Por exemplo, o número 0 (zero) como opção no *menu*.
O(A) estudante terá liberdade para escolher como implementar a funcionalidade de *menu*.

Observações

1. Uma `data` é formada por seu `dia`, `mes` e `ano`;
2. Considere que as datas a serem aplicadas ao sistema serão, sempre, no intervalo de 01/01/1900 a 31/12/2200;
3. Fique atento a um importante evento que ocorreu no mês de outubro de 1582 envolvendo o calendário Gregoriano – pesquise sobre isto antes de implementar todas as funções. Em partircular a função `obtemDiaSemanaData (Data d);`
4. `TRUE` = 1; `FALSE` = 0;
5. A função `comparaData (Data d1, Data d2)` deve retornar:

   `MENOR` quando d1 < d2;

   `IGUAL` quando d1 = d2;

   `MAIOR` quando d1 > d2.

   com `MENOR` = -1; `IGUAL` = 0 e `MAIOR` = 1.