Instituto de Computação - Unicamp

MC102 - Algoritmos e Programação de Computadores

Laboratório 16 - Conjuntos Dinâmicos

Prazo de entrega: **02/06/2017 23:59:59**

Peso: **1**

Professor: Guido Araújo

Monitor: Arthur Pratti Dadalto
Monitor: Cristina Cavalcante
Monitor: Luís Felipe Mattos
Monitor: Paulo Finardi
Monitor: Pedro Alves

Descrição

Como visto no Lab13, a Teoria dos Conjuntos é um ramo da matemática muito utilizada na computação e em diversas outras áreas.

Neste laboratório você deverá implementar funções que realizam operações sobre conjuntos *com tamanho máximo indeterminado*.

As operações a serem realizadas sobre conjuntos são:

- Dado um conjunto A, se x é um de seus elementos então dizemos que x **pertence** ao conjunto A, e denotamos isto por $x \in A$. Caso x não pertença ao conjunto A, denotamos isto por $x \notin A$.
- Dados dois conjuntos A e B, se cada um dos elementos de B pertencer também a A, então dizemos que A **contém** B, ou de forma equivalente B **está contido** em A, e denotamos isto por $B \subseteq A$. Caso

B não esteja contido em A, denotamos isto por $B \not\subset A$.

- A operação de **união** de dois conjuntos A e B, denotada por $A \cup B$, tem como resultado um outro conjunto contendo os elementos que estão em A ou B. Por exemplo, se $A = \{a,b,c\}$ e $B = \{b,c,d\}$, a operação $A \cup B$ terá como resultado o conjunto $\{a,b,c,d\}$.

- A operação de **interseção** de dois conjuntos A e B, denotada por $A \cap B$, tem como resultado um outro conjunto contendo os elementos que estão em ambos conjuntos A e B. Por exemplo, se $A = \{a,b,c\}$ e $B = \{b,c,d\}$, sua interseção $A \cap B$ será $\{b,c\}$.

- A operação **diferença** do conjunto A para o conjunto B, denotada por $A \setminus B$, tem como resultado um conjunto contendo os elementos que estão em A mas não estão em B. Por exemplo, se $A = \{a,b,c\}$ e $B = \{b,c,d\}$, a diferença $A \setminus B$, será $\{a\}$, enquanto que a diferença $B \setminus A$ será $\{d\}$.

Como o tamanho máximo dos conjuntos são indeterminados, usaremos alocação dinâmica para alterar o tamanho dos conjuntos em tempo de execução. Os conjuntos serão representados por um vetor e dois inteiros, `tamanho` e `capacidade`. O `tamanho` indica a quantidade de elementos que o conjunto contém e `capacidade` indica o tamanho do vetor alocado para o conjunto. Desta forma o conjunto consegue armazenar até `capacidade` elementos. Se o vetor ficar cheio e novos elementos precisarem ser inseridos no conjunto temos que realocá-lo para um vetor maior. De forma similar se muitos elementos forem removidos de um conjunto, o vetor alocado deverá ser diminído para ocupar menos espaço em memória.

O objetivo deste laboratório é criar uma biblioteca de funções em C para realizar operações sobre conjuntos de números inteiros. Os conjuntos serão representados utilizando-se vetores.

Você deve implementar funções que realizam as seguintes operações:

- **Ordena:** ordena o conjunto especificado de maneira crescente.

- **Pertence:** verifica se um elemento pertence ao conjunto especificado, retornando verdadeiro ou falso.

- **Continência:** verifica se um conjunto está contido em outro

conjunto retornando verdadeiro ou falso.

- **Inicialização:** inicializa um conjunto com tamanho zero e capacidade dois (02).
- **Adição:** adiciona um elemento em um conjunto, alterando o conjunto com a adição do novo elemento *caso ele já não pertença ao mesmo*.
    - Cada conjunto apresenta um tamanho atual e a capacidade que devem ser atualizados a medida que novos elementos forem adicionados.
    - Caso a quantidade de elementos do conjunto (tamanho) seja igual a capacidade do vetor alocado e um novo elemento tenha que ser inserido, então o vetor do conjunto terá de ser realocado com o dobro de sua capacidade atual. (Note que você deve realocar o vetor para obter mais memória).
- **Subtração:** remove um elemento de um conjunto, alterando o conjunto especificado com a remoção do elemento *caso ele pertença ao conjunto*.
    - Cada conjunto apresenta um tamanho atual e a capacidade que devem ser atualizados a medida que elementos forem removidos.
    - Caso a capacidade do conjunto seja maior que dois (02) e a quantidade de elementos do conjunto seja menor ou igual que 1/4 da sua capacidade, então o vetor deverá ser realocado com a metade de sua capacidade (Note que você deve realocar o vetor para reduzir o consumo de memória).
- **União:** faz a união de dois conjuntos, criando um conjunto com os elementos dos dois conjuntos de entrada.
- **Interseção:** faz a interseção de dois conjuntos, criando um conjunto com os elementos que pertencem aos dois conjuntos de entrada.
- **Diferença:** Faz a diferença de dois conjuntos, criando um conjunto com os elementos do primeiro que não se encontram no segundo.

OBS: Os vetores A e B são criados na função `main` que é fornecida

neste laboratório no arquivo `lab16_main.c`. Vocês devem apenas implementar e submeter o arquivo `lab16.c`, com as funções descritas abaixo.

Funções

Observações gerais:

- Os conjuntos armazenam apenas números inteiros.
- O tamanho atual e a capacidade de cada conjunto é passado por parâmetro.

A descrição geral dos parâmetros de entrada e saída das funções está descrita nos comentários dos protótipos das funções, que são fornecidos a seguir:

Linguagem C:

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

/*
---------------------------------------------------------------------
Aluno(a):
RA:
---------------------------------------------------------------------
*/


/*
---------------------------------------------------------------------
void ordena(int *conj, int tam);


Esta funcao deve ordenar um conjunto dado como parametro.

Parametros:

- conj -> Ponteiro para o conjunto;
- tam -> Quantidade de elementos do conjunto;

---------------------------------------------------------------------
*/
```

```
void ordena(int *conj, int tam) {

}

/*
-----------------------------------------------------------------
int pertence(int *conj, int tam, int elemento);


Esta funcao deve verificar se um elemento esta presente no conjur

Parametros:

- conj -> Ponteiro para o conjunto;
- tam -> Quantidade de elementos do conjunto;
- elemento -> Elemento no qual deve ser ser verificado se esta pr

Retorno

- 1 Caso o elemento PERTENCA conjunto;
- 0 Caso o elemento NAO PERTENCA ao conjunto;

-----------------------------------------------------------------
*/

int pertence(int *conj, int tam, int elemento) {
  return 1;
}

/*
-----------------------------------------------------------------
int contido(int *conj_A, int *conj_B, int tam_A, int tam_B);


Esta funcao deve verificar se o conjunto A esta contido no conjur

Parametros:
- conj_A -> Ponteiro para o conjunto A;
- conj_B -> Ponteiro para o conjunto B;
- tam_A -> Quantidade de elementos do conjunto A;
- tam_B -> Quantidade de elementos do conjunto B;

Retorno

- 1 Caso o conjunto A ESTEJA CONTIDO no conjunto B;
- 0 Caso o conjunto A NAO ESTEJA CONTIDO no conjunto B;
```

```
-----------------------------------------------------------------
*/

int contido(int *conj_A, int *conj_B, int tam_A, int tam_B) {
  return 1;
}

/*
-----------------------------------------------------------------
int* init(int *tam, int *cap);


Esta funcao deve inicializar um vetor(conjunto).

Parametros:
- tam -> Ponteiro para a quantidade de elementos do conjunto;
- cap -> Ponteiro para a capacidade de elementos do conjunto;

OBS:
- A capacidade inicial do vetor(conjunto) deve ser 2.
- O quantidade inicial de elementos no vetor(conjunto) deve ser z
- Nao confundir capacidade com quantidade de elementos.

Retorno

- Ponteiro para o conjunto;

-----------------------------------------------------------------
*/

int* init(int *tam, int *cap) {
  return NULL;
}

/*
-----------------------------------------------------------------
int* adicao(int *conj, int *tam, int *cap, int elemento);


Esta funcao deve adicionar um novo elemento no conjunto, ou seja,
conjunto o mesmo NAO deve ser adicionado.

Parametros:
- conj -> Ponteiro para o conjunto;
- tam -> Ponteiro para a quantidade de elementos do conjunto;
- cap -> Ponteiro para a capacidade de elementos do conjunto;
```

```
- elemento -> Elementos para ser adicionado;

OBS:
- Ao adicionar um novo elemento o tamanho atual do conjunto dever
- Caso o ponteiro para o conjunto seja NULL, o conjunto devera se
e a atualizacao da capacidade deverar ser feita;
- Caso a quantidade de elementos do conjunto seja igual a capacic
ser inserido, então o conjunto tera de ser realocado com o dobro
devera ser atualizada;

Retorno

- Ponteiro para o conjunto;

-----------------------------------------------------------------
*/

int* adicao(int *conj, int *tam, int *cap, int elemento) {
  return NULL;
}

/*
-----------------------------------------------------------------
int* subtracao(int *conj, int *tam, int *cap, int elemento);


Esta funcao deve remover um elemento no conjunto caso ele exista.

Parametros:
- conj -> Ponteiro para o conjunto;
- tam -> Ponteiro para a quantidade de elementos do conjunto;
- cap -> Ponteiro para a capacidade de elementos do conjunto;
- elemento -> Elementos para ser removido;

OBS:
- Ao remover um elemento o tamanho atual do conjunto devera ser a
- Caso a capacidade seja maior que dois (02) e a quantidade de el
o conjunto tera de ser realocado com a metade da capacidade e a c

Retorno

- Ponteiro para o conjunto;

-----------------------------------------------------------------
*/

int* subtracao(int *conj, int *tam, int *cap, int elemento) {
```

```
  return NULL;
}

/*
-----------------------------------------------------------------
int* uniao(int *conj_A, int *conj_B, int tam_A, int tam_B, int *t


Esta funcao deve computar a uniao entre os conjuntos A e B. O res
um novo conjunto C.

Parametros:
- conj_A -> Ponteiro para o conjunto A;
- conj_b -> Ponteiro para o conjunto B;
- tam_A -> Quantidade de elementos do conjunto A;
- tam_B -> Quantidade de elementos do conjunto B;
- tam_C -> Ponteiro para a quantidade de elementos do conjunto re
- cap_C -> Ponteiro para a capacidade de elementos do conjunto re
- elemento -> Elementos para ser removido;

OBS:
- O tamanho atual e a capacidade do conjunto resultante C devera
init e adicao.
- Os valores de quantidade de elementos e capacidade do conjunto
respectivamente nos parametros tam_C e cap_C.

Retorno

- Ponteiro para o conjunto C;

-----------------------------------------------------------------
*/

int* uniao(int *conj_A, int *conj_B, int tam_A, int tam_B, int *t
  return NULL;
}

/*
-----------------------------------------------------------------
int* intersecao(int *conj_A, int *conj_B, int tam_A, int tam_B, i


Esta funcao deve computar a intersecao entre os conjuntos A e B.
armazenada em um novo conjunto C.

Parametros:
- conj_A -> Ponteiro para o conjunto A;
```

```
- conj_b -> Ponteiro para o conjunto B;
- tam_A -> Quantidade de elementos do conjunto A;
- tam_B -> Quantidade de elementos do conjunto B;
- tam_C -> Ponteiro para a quantidade de elementos do conjunto re
- cap_C -> Ponteiro para a capacidade de elementos do conjunto re
- elemento -> Elementos para ser removido;

OBS:
- O tamanho atual e a capacidade do conjunto resultante C devera
init e adicao.
- Os valores de quantidade de elementos e capacidade do conjunto
respectivamente nos parametros tam_C e cap_C.

Retorno

- Ponteiro para o conjunto C;

-----------------------------------------------------------------
*/

int* intersecao(int *conj_A, int *conj_B, int tam_A, int tam_B, i
  return NULL;
}

/*
-----------------------------------------------------------------
int* diferenca(int *conj_A, int *conj_B, int tam_A, int tam_B, ir


Esta funcao deve computar a diferenca entre os conjuntos A e B. C
armazenada em um novo conjunto C.

Parametros:
- conj_A -> Ponteiro para o conjunto A;
- conj_b -> Ponteiro para o conjunto B;
- tam_A -> Quantidade de elementos do conjunto A;
- tam_B -> Quantidade de elementos do conjunto B;
- tam_C -> Ponteiro para a quantidade de elementos do conjunto re
- cap_C -> Ponteiro para a capacidade de elementos do conjunto re
- elemento -> Elementos para ser removido;

OBS:
- O tamanho atual e a capacidade do conjunto resultante C devera
init e adicao.
- Os valores de quantidade de elementos e capacidade do conjunto
respectivamente nos parametros tam_C e cap_C.
```

```
Retorno

- Ponteiro para o conjunto C;

-----------------------------------------------------------------
*/

int* diferenca(int *conj_A, int *conj_B, int tam_A, int tam_B, ir
  return NULL;
}
```

Múltiplos Arquivos e Função Principal

Neste laboratório vamos utilizar o conceito de dividir o código em múltiplos arquivos. Quando se implementa programas grandes é comum separar o código em vários arquivos com a extensão .c, onde cada arquivo implementa um conjunto de funções relacionadas entre si. Isto facilita a manutenção e a leitura do código. Para compilar um código organizado dessa forma, basta passar todos os arquivos na linha de comando para o compilador.

Para esse laboratório você só deverá implementar as funções descritas acima. A função principal (**main**) será fornecida em um arquivo separado, chamado `lab16_main.c` .
Um link para ele também está disponível na página da tarefa.

Vamos ao exemplo de como compilar o seu programa em C. Até agora, a forma que utilizamos (de forma simplificada) era a seguinte:

```
gcc -o labXX labXX.c
```

Nesse laboratório, no entanto, para compilar o seu programa basta adicionar o arquivo extra que provemos (lab16_main.c) ao final da linha de comando, como no exemplo a seguir:

```
gcc -o lab16 lab16.c lab16_main.c
```

OBS: A linha completa de compilação para esse laboratório pode ser vista na sessão de Observações.

A organização do conteúdo de cada arquivo é a seguinte:

- `lab16` :
  - funções auxiliares que você queira escrever;
  - `ordena(...)` .
  - `pertence(...)` ;
  - `contido(...)` ;
  - `init(...)` ;
  - `adicao(...)` ;
  - `subtracao(...)` ;
  - `uniao(...)` ;
  - `intersecao(...)` ;
  - `diferenca(...)` ;
- `lab16_main` :
  - funções auxiliares para a main;
  - `main()` .

Também está disponível um protótipo do arquivo que você deve submeter ao *SuSy* (lab16.c). Esse arquivo e o arquivo auxiliar (lab16_main.c) também podem ser encontrados na página da tarefa:

- `lab16.c`
- `lab16_main.c`

Reforçando

Neste laboratório você não precisará se preocupar em ler a entrada a partir da entrada padrão, nem em escrever a saída. Seu trabalho é apenas implementar as funções descritas. A função `main()` que é fornecida no arquivo `lab16_main.c` se encarrega dessa parte.

Você também **não deve** submeter o arquivo `lab16_main.c` para o *SuSy*, somente o arquivo `lab16.c` .

As sessões abaixo, de Entrada e Saída, descrevem os formatos de entrada e saída, mas você não precisa se preocupar com eles.

Entrada

A entrada consiste de operações a serem realizadas sobre dois conjuntos nomeados de A e B. Os conjuntos iniciam vazios e cada linha da entrada descreve uma operação a ser realizada sobre um ou entre os dois conjuntos.

As operações são:

- `C = {x1, x2, x3, ... ,xn}` : substitui o conteúdo do conjunto `C` com os `N` elementos.
- `x e C` : verifica se o elemento `x` pertence ao conjunto `C` .
- `C1 c C2` : verifica se o conjunto `C1` está contido no conjunto `C2` .
- `C += x` : adiciona o elemento `x` ao conjunto `C` .
- `C -= x` : remove o elemento `x` do conjunto `C` .
- `C1 u C2` : exibe a união entre `C1` e `C2` .
- `C1 ^ C2` : exibe a interseção entre `C1` e `C2` .
- `C1 \ C2` : exibe a diferença entre `C1` e `C2` .
- `Q` : Encerra a execução do programa

Onde:

- `C` é um dos conjuntos `A` ou `B`
- `C1` e `C2` são conjuntos distintos `A` e `B` em qualquer ordem.
- `x` é um elemento
- `x1, x2, x3, ... , xn` são `N` elementos.
- `Q` é a letra Q

Saída

Cada linha da saída do programa contém o resultado da execução de cada operação dada na entrada, de forma que a saída possui uma linha a menos que a quantidade de linhas da entrada.

O retorno das operações podem ter um dos 3 formatos distintos:

- Para as operações `C` , `C = {x1, x2, x3, ... ,xn}` , `C += x` , ou `C -= x` :

Imprime o conteúdo do conjunto C, com os elementos em ordem crescente.

Formato da saída: `C = {x1, x2, x3, ..., xn}, tamanho = tam, capacidade = cap`

- para as operações `x e C` ou `C1 c C2`:

  Imprime Verdadeiro ou Falso.

  Formato: `verdadeiro` ou `falso`

- Para as operações `C1 u C2`, `C1 ^ C2` ou `C1 \ C2`:

  Imprime o conjunto resultado da operação, com os elementos em ordem crescente.

  Formato: `C1 op C2 = {x1, x2, x3, ..., xn}, tamanho = tam, capacidade = cap`

- Para a operação `Q` nada é impresso. Se encerra a execução do programa.

Exemplos

Teste 01

Entrada

```
A = {707}
B = {-586, -509, -235, 181, 995}
B += -509
A u B
B ^ A
A ^ B
B ^ A
-586 e B
-55 e A
B \ A
B c A
B u A
Q
```

Saída

```
A = {707}, tamanho = 1, capacidade = 2
B = {-586, -509, -235, 181, 995}, tamanho = 5, capacidade = 8
B = {-586, -509, -235, 181, 995}, tamanho = 5, capacidade = 8
A u B = {-586, -509, -235, 181, 707, 995}, tamanho = 6, capacidac
B ^ A = {}, tamanho = 0, capacidade = 2
A ^ B = {}, tamanho = 0, capacidade = 2
B ^ A = {}, tamanho = 0, capacidade = 2
verdadeiro
falso
B \ A = {-586, -509, -235, 181, 995}, tamanho = 5, capacidade = 8
falso
B u A = {-586, -509, -235, 181, 707, 995}, tamanho = 6, capacidac
```

Teste 02

Entrada

```
A = {-774, -728, -705, 349}
B = {135}
B c A
B \ A
A u B
-69 e B
B -= 89
B += 687
B u A
A \ B
A += -28
A u B
B ^ A
B -= 135
A c B
B u A
A += 349
A += 39
B \ A
B c A
A -= 39
A -= -935
Q
```

Saída

```
A = {-774, -728, -705, 349}, tamanho = 4, capacidade = 4
B = {135}, tamanho = 1, capacidade = 2
falso
B \ A = {135}, tamanho = 1, capacidade = 2
A u B = {-774, -728, -705, 135, 349}, tamanho = 5, capacidade = 8
falso
B = {135}, tamanho = 1, capacidade = 2
B = {135, 687}, tamanho = 2, capacidade = 2
B u A = {-774, -728, -705, 135, 349, 687}, tamanho = 6, capacidac
A \ B = {-774, -728, -705, 349}, tamanho = 4, capacidade = 4
A = {-774, -728, -705, -28, 349}, tamanho = 5, capacidade = 8
A u B = {-774, -728, -705, -28, 135, 349, 687}, tamanho = 7, capa
B ^ A = {}, tamanho = 0, capacidade = 2
B = {687}, tamanho = 1, capacidade = 2
falso
B u A = {-774, -728, -705, -28, 349, 687}, tamanho = 6, capacidac
A = {-774, -728, -705, -28, 349}, tamanho = 5, capacidade = 8
A = {-774, -728, -705, -28, 39, 349}, tamanho = 6, capacidade = 8
B \ A = {687}, tamanho = 1, capacidade = 2
falso
A = {-774, -728, -705, -28, 349}, tamanho = 5, capacidade = 8
A = {-774, -728, -705, -28, 349}, tamanho = 5, capacidade = 8
```

Teste 03

Entrada

```
A = {-475, 271, 436, 866}
B = {-604, 677}
B c A
515 e B
B += -584
B u A
A c B
A ^ B
B c A
B ^ A
A -= 436
815 e A
B \ A
A -= -894
A \ B
B ^ A
A += -475
```

```
-118 e B
B ^ A
B ^ A
B u A
B \ A
935 e B
A ^ B
A \ B
A c B
A \ B
A += 271
B ^ A
A ^ B
B \ A
677 e B
Q
```

Saída

```
A = {-475, 271, 436, 866}, tamanho = 4, capacidade = 4
B = {-604, 677}, tamanho = 2, capacidade = 2
falso
falso
B = {-604, -584, 677}, tamanho = 3, capacidade = 4
B u A = {-604, -584, -475, 271, 436, 677, 866}, tamanho = 7, capa
falso
A ^ B = {}, tamanho = 0, capacidade = 2
falso
B ^ A = {}, tamanho = 0, capacidade = 2
A = {-475, 271, 866}, tamanho = 3, capacidade = 4
falso
B \ A = {-604, -584, 677}, tamanho = 3, capacidade = 4
A = {-475, 271, 866}, tamanho = 3, capacidade = 4
A \ B = {-475, 271, 866}, tamanho = 3, capacidade = 4
B ^ A = {}, tamanho = 0, capacidade = 2
A = {-475, 271, 866}, tamanho = 3, capacidade = 4
falso
B ^ A = {}, tamanho = 0, capacidade = 2
B ^ A = {}, tamanho = 0, capacidade = 2
B u A = {-604, -584, -475, 271, 677, 866}, tamanho = 6, capacidac
B \ A = {-604, -584, 677}, tamanho = 3, capacidade = 4
falso
A ^ B = {}, tamanho = 0, capacidade = 2
A \ B = {-475, 271, 866}, tamanho = 3, capacidade = 4
falso
```

```
A \ B = {-475, 271, 866}, tamanho = 3, capacidade = 4
A = {-475, 271, 866}, tamanho = 3, capacidade = 4
B ^ A = {}, tamanho = 0, capacidade = 2
A ^ B = {}, tamanho = 0, capacidade = 2
B \ A = {-604, -584, 677}, tamanho = 3, capacidade = 4
verdadeiro
```

Para mais exemplos, consulte os testes abertos no Susy.

Observações

- Você **não deve** submeter o arquivo `lab16_main.c` para o *SuSy*, somente o arquivo `lab16.c`.
- O número máximo de submissões é **15**.
- Para a realização dos testes do SuSy, a compilação dos programas desenvolvidos em C irá considerar o comando:
  `gcc -std=c99 -pedantic -Wall -o lab16 lab16.c lab16_main.c`.
- Você deve incluir, no início do seu programa, uma breve descrição dos objetivos do programa, da entrada e da saída, além do seu nome e do seu RA.
- Indente corretamente o seu código e inclua comentários no decorrer do seu programa.

Critérios importantes

Independentemente dos resultados dos testes do *SuSy*, o não cumprimento dos critérios abaixo implicará em nota zero nesta tarefa de laboratório.

- Os únicos headers aceitos para essa tarefa são `stdio.h` e `stdlib.h`.