Instituto de Computação - Unicamp

MC102 - Algoritmos e Programação de Computadores

Laboratório 13 - Conjuntos

Prazo de entrega: **05/05/2017 23:59:59**

Peso: **1**

Professor: Eduardo C. Xavier
Professor: Guido Araújo

Monitor: Arthur Pratti Dadalto
Monitor: Cristina Cavalcante
Monitor: Klairton de Lima Brito
Monitor: Luís Felipe Mattos
Monitor: Paulo Finardi
Monitor: Paulo Lucas Rodrigues Lacerda
Monitor: Pedro Alves
Monitor: Renan Vilas Novas
Monitor: Vinicius de Novaes Guimarães Pereira

>

Descrição

A Teoria dos Conjuntos é um ramo da matemática que estuda propriedades de coleções de objetos, e é utilizada para melhor entendermos e modelarmos diversos problemas. Um conjunto é uma coleção de elementos distintos. Por exemplo, as letras *c*, *o* e *p* são elementos distintos quando considerados isoladamente, mas quando considerados coletivamente formam o conjunto *{c,o,p}* e, nesse caso, *c*, *o* e

p são os elementos do conjunto. A cardinalidade ou tamanho de um conjunto é o número de elementos desse conjunto. Note que um conjunto não pode ter mais de um elemento do mesmo valor. Por exemplo *{a,b,a,b,c}* não é um conjunto, mas *{a,b,c}* sim.

Abaixo temos uma lista de definições e operações que podem ser realizadas sobre conjuntos:

- 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}*.

- Considerando como *U* o conjunto universo, o **complemento** do conjunto *A* , denotada por ~A, tem como resultado um conjunto contendo os elementos que não estão em *A* mas que estão em *U*. Por exemplo, se *U* é o conjunto de todas as letras minúsculas do alfabeto inglês, e *A* = *{a,b,c}*, o complemento *~A* será $U \setminus A$.

O objetivo deste laboratório é criar uma biblioteca de funções em C para realizar operações sobre o conjunto das letras minúsculas do alfabeto inglês. Os conjuntos serão representados utilizando-se vetores.

Você deve implementar funções que realizam as seguintes operações:

- **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.
- **Adição:** adiciona um elemento em um conjunto, alterando o conjunto com a adição do novo elemento *caso ele já não pertença ao mesmo*.
- **Subtração:** remove um elemento de um conjunto, alterando o conjunto especificado com a remoção do elemento *caso ele pertença ao conjunto*.
- **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.
- **Complemento:** Faz o complemento de um conjunto, criando um conjunto com os elementos que estão em *U* mas não estão no primeiro conjunto.

OBS: Os vetores são criados na função `main` que é fornecida neste laboratório no arquivo `lab15_main.c`. Os vetores são criados com tamanho 26. Vocês devem apenas implementar e submeter o arquivo `lab15.c`, com as funções descritas abaixo.

Funções

Observações gerais:

- Os conjuntos armazenam apenas letras minúsculas do alfabeto inglês: a, b, c, k, w, y, etc.
- O Conjunto universo *U* é o conjuntos de todas as letras minúsculas do alfabeto inglês.
- O tamanho atual de cada conjunto é passado por parâmetro.
- Os vetores que armazenam os conjuntos não estão e não precisam estar ordenados.

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:

```
/* Laboratorio 15 - Conjuntos
 * Nome:
 * RA:
 */

#include <stdio.h>

/* Funcao: pertence
 *
 * Parametros:
 *   conj: vetor contendo o conjunto de entrada
 *    tam: tamanho do conjunto
 *  letra: elemento a ser verificado pertinencia
 *
 * Retorno:
 *   1 se letra pertence a conj e 0 caso contrario
 */
int pertence(char conj[], int tam, char letra){

  /* Implementar a funcao e trocar o valor de retorno */
  return 0;
}

/* Funcao: contido
```

```
 *
 * Parametros:
 *   conj1: vetor contendo um conjunto de entrada
 *   conj2: vetor contendo um conjunto de entrada
 *    tam1: tamanho do conjunto conj1
 *    tam2: tamanho do conjunto conj2
 *
 * Retorno:
 *   1 se conj1 esta contido em conj2 e 0 caso contrario
 */
int contido(char conj1[], char conj2[], int tam1, int tam2){
  /* Implementar a funcao e trocar o valor de retorno */
  return 0;
}

/* Funcoes: adicao e subtracao
 *
 * Parametros:
 *   conj: vetor contendo o conjunto que tera incluso ou removido o e
 *    tam: tamanho do conjunto
 *  letra: elemento a ser adicionado ou removido
 *
 * Retorno:
 *   tamanho do conjunto apos a operacao.
 */
int adicao(char conj[], int tam, char letra){
  /* Implementar a funcao e trocar o valor de retorno */
  return 0;
}

int subtracao(char conj[], int tam, char letra){
  /* Implementar a funcao e trocar o valor de retorno */
  return 0;
}

/* Funcoes: uniao, intersecao e diferenca
 *
 * Parametros:
 *   conjRes: vetor contendo o conjunto de saida/resultado da operaca
 *     conj1: vetor contendo o conjunto de entrada do primeiro operan
 *     conj2: vetor contendo o conjunto de entrada do segundo operand
 *      tam1: tamanho do conjunto conj1
 *      tam2: tamanho do conjunto conj2
```

```
 *
 * Retorno:
 *   tamanho do conjunto de saida conjRes.
 */
int uniao(char destRes[], char conj1[], char conj2[], int tam1, int t
  /* Implementar a funcao e trocar o valor de retorno */
  return 0;
}

int intersecao(char destRes[], char conj1[], char conj2[], int tam1,
  /* Implementar a funcao e trocar o valor de retorno */
  return 0;
}

int diferenca(char destRes[], char conj1[], char conj2[], int tam1, i
  /* Implementar a funcao e trocar o valor de retorno */
  return 0;
}


/* Funcao: complemento
 * Guarda em conjRes o resultado da operação U-conj, onde U é o conju
 * de todas as letras minusculas do alfabeto ingles
 *
 * Parametros:
 *   conjRes: vetor contendo o conjunto de saida/resultado da operaca
 *      conj: vetor contendo o conjunto de entrada do primeiro operan
 *       tam: tamanho do conjunto conj
 *
 * Retorno:
 *   tamanho do conjunto de saida conjRes.
 */
int complemento(char conjRes[], char conj[], int tam){
  /* Implementar a funcao e trocar o valor de retorno */
  return 0;
}
```

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 `lab13_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 (lab13_main.c) ao final da linha de comando, como no exemplo a seguir:

```
gcc -o lab13 lab13.c lab13_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:

- `lab13` :
  - funções auxiliares que você queira escrever;
  - `pertence(...)` ;
  - `contido(...)` ;
  - `adicao(...)` ;
  - `subtracao(...)` ;
  - `uniao(...)` ;
  - `intersecao(...)` ;

    - `diferenca(...)`;
    - `complemento(...)`.

- `lab13_main`:
    - funções auxiliares para a main;
    - `main()`.

Também está disponível um protótipo do arquivo que você deve submeter ao *SuSy* (lab13.c). Esse arquivo e o arquivo auxiliar (lab13_main.c) também podem ser encontrados na página da tarefa:

- `lab13.c`
- `lab13_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 `lab13_main.c` se encarrega dessa parte.

Você também **não deve** submeter o arquivo `lab13_main.c` para o *SuSy*, somente o arquivo `lab13.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` : Não faça nada no conjunto, Só imprima o seu conteúdo
- `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` .
- `-C` : exibe o complemento do conjunto `C` .
- `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}`

- 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}`

- Para a operação `-C`:

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

  Formato: `-C = {x1, x2, x3, ..., xn}`

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

Exemplos

Teste 01

Entrada

```
A += l
A += c
A += f
A += c
l e A
b e A
Q
```

Saída

```
A = {l}
A = {c, l}
A = {c, f, l}
A = {c, f, l}
verdadeiro
falso
```

Teste 02

Entrada

```
A = {c, f, h}
A -= f
A -= c
A -= f
A
h e A
f e A
Q
```

Saída

```
A = {c, f, h}
A = {c, h}
A = {h}
A = {h}
A = {h}
verdadeiro
falso
```

Teste 03

Entrada

```
A = {c, f, h}
```

```
B = {c, f}
B c A
A c B
B += d
B c A
Q
```

Saída

```
A = {c, f, h}
B = {c, f}
verdadeiro
falso
B = {c, d, f}
falso
```

Teste 04

Entrada

```
A = {c, f, h}
B = {}
B c A
A c B
B = {f}
B c A
B = {b, f}
B c A
A c B
Q
```

Saída

```
A = {c, f, h}
B = {}
verdadeiro
falso
B = {f}
verdadeiro
```

```
B = {b, f}
falso
falso
```

Teste 06

Entrada

```
A = {c, f, h}
B = {c, d, e}
A ^ B
A = {b, c, d}
B = {e, f, g}
A ^ B
Q
```

Saída

```
A = {c, f, h}
B = {c, d, e}
A ^ B = {c}
A = {b, c, d}
B = {e, f, g}
A ^ B = {}
```

Teste 07

Entrada

```
A = {c, f, h}
B = {c, d, e}
A \ B
B \ A
-A
-B
A = {}
A \ B
B \ A
```

```
-A
-B
Q
```

Saída

```
A = {c, f, h}
B = {c, d, e}
A \ B = {f, h}
B \ A = {d, e}
-A = {a, b, d, e, g, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x,
-B = {a, b, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x,
A = {}
A \ B = {}
B \ A = {c, d, e}
-A = {a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u,
-B = {a, b, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x,
```

Para mais exemplos, consulte os testes abertos no Susy.

Observações

- Você **não deve** submeter o arquivo `lab13_main.c` para o *SuSy*, somente o arquivo `lab13.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 lab13 lab13.c lab13_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.

- O único header aceito para essa tarefa é o `stdio.h`.