**Questão 01) Quicksort**
Considere o código abaixo para o algoritmo *quicksort*:

```
void quicksort(int * v, int p, int r) {
      int q;
      if (p < r) {
            q = partition (v, p, r);
            quicksort (v, p, q-1);
            quicksort (v, q+1, r);
      }
}
```

Escreva um programa que ordena um vetor de números inteiros usando o *quicksort*, seguindo algoritmo acima. A chamada externa (a partir da função **main**) é **quicksort (vetor, 1, n)**.

Entrada: a primeira linha recebe o número inteiro **n** (n>0), referente a quantidade de número a ordenar; a segunda linha recebe **n** números inteiros, separados por um espaço em branco entre si.
Saída: uma única linha com os números ordenados, separados por um espaço em branco entre si.

Exemplo

| Entrada | Saída |
|---|---|
| **5**<br>**7 1 3 5 2** | **1 2 3 5 7** |

**Questão 02) TAD Lista por encadeamento simples**
Considere abaixo as definições das estruturas de dados para um **TAD Lista** simplificado (lista linear de números inteiros), visando à sua implementação por encadeamento simples:

```
typedef struct lista Lista;
typedef struct elemento Elemento;
struct elemento {
      int elemento;
      Elemento * proximo;
};
struct lista {
      Elemento * cabeca;
      int tamanho;
};
```

Seja abaixo a definição das operações para esse **TAD Lista**:

```
Lista * lista_cria ();
void lista_libera (Lista * p);
int lista_insere_posicao (Lista * p, int elemento, int posicao);
int lista_obtem_elemento  (Lista * p, int posicao);
int lista_obtem_tamanho  (Lista * p);
```

Escreva um programa que cria uma lista linear, conforme as definições acima, cuja inserção de elementos sempre ocorre no meio da lista.
Muito importante: O programa (função **main**) usará somente as operações previstas para o TAD.
A seguir estão ilustrados alguns casos para a inserção de elementos.

| Lista antes da inserção | Tamanho da lista | Posição para inserir | Lista após a inserção |
|---|---|---|---|
| | 0 | 1 | **99** |
| 11 | 1 | 1 | **99** 11 |
| 11 22 | 2 | 2 | 11 **99** 22 |
| 11 22 33 | 3 | 2 | 11 **99** 22 33 |
| 11 22 33 44 | 4 | 3 | 11 22 **99** 33 44 |
| 11 22 33 44 55 | 5 | 3 | 11 22 **99** 33 44 55 |

Entrada: A primeira linha contém um número **n** (n>0), que indica a quantidade de números a inserir na lista. A segunda linha possui os elementos a inserir, separados por um espaço em branco.
Saída: O conteúdo da lista após as inserções; os elementos são separados por um espaço em branco.
Exemplo

| Entrada | Saída |
|---|---|
| **5**<br>**7 1 3 5 2** | **1 5 2 3 7** |

**Questão 03) TAD Vetor**
Considere abaixo as definições das estruturas de dados para um **TAD Vetor** de números inteiros, bem como a definição das operações para esse **TAD Vetor**:

```
typedef struct vetor Vetor;
struct vetor {
      int * numeros;
      int tamanho;
      int tamanho_maximo;
};
```

```
Vetor * vetor_cria (int tamanho_maximo);
void vetor_libera (Vetor * p);
void vetor_insere (Vetor * p, int numero);
void vetor_remove (Vetor * p, int numero);
int vetor_obtem_minimo (Vetor * p);
int vetor_obtem_tamanho (Vetor * p);
```

Escreva um programa que cria um vetor, conforme as definições acima.
Muito importante: O programa (função **main**) usará somente as operações previstas para o TAD.

Entrada: A primeira linha contém um número **m** (m>0), que indica a quantidade máxima de números inteiros no vetor. Cada das demais linhas possui uma das seguintes operações:

- **I num**, para a inserção do **num** no vetor; se **num** já existir no vetor ou vetor não tenha alcançado a quantidade máxima, a inserção não será realizada;
- **R num**, para a remoção do **num** do vetor, caso **num** esteja no vetor;
- **M**, para imprimir o menor elemento do vetor; se o vetor estiver vazio, imprimir -1;
- **F**, para finalizar o programa.

Saída: uma linha em resposta a cada operação do tipo **M**.
Exemplo

| Entrada | Saída |
|---|---|
| **100**<br>**I 1**<br>**I 22**<br>**I 1**<br>**I 13**<br>**M**<br>**R 1**<br>**M**<br>**R 13**<br>**R 44**<br>**M**<br>**F** | **1**<br>**13**<br>**22** |

**Questão 04) Mergesort**
Considere o código abaixo para o algoritmo *mergesort*:

```
void mergesort (int * v, int p, int r) {
      int q;
      if (p < r)  {
      q = (p + r) / 2;
      mergesort (v, p, q);
      mergesort (v, q+1, r);
      merge (v, p, q, r);
      }
}
```

Escreva um programa que ordena um vetor de números inteiros usando o *mergesort*, seguindo algoritmo acima. A chamada externa (a partir da função **main**) é **mergesort(vetor, 1, n)**.

Entrada: a primeira linha recebe o número inteiro **n** (n>0), referente a quantidade de números a ordenar; a segunda linha recebe **n** números inteiros, separados por um espaço em branco entre si.
Saída: uma única linha com os números ordenados, separados por um espaço em branco entre si.
Exemplo

| Entrada | Saída |
|---|---|
| **5**<br>**7 1 3 5 2** | **1 2 3 5 7** |

**Questão 05) TAD Lista por encadeamento simples**
Considere abaixo as definições das estruturas de dados para um **TAD Lista** simplificado (lista linear de números inteiros), visando à sua implementação por encadeamento simples:

```
typedef struct lista Lista;
typedef struct elemento Elemento;
struct elemento {
      int elemento;
      Elemento * proximo;
};
struct lista {
      Elemento * cabeca;
      int tamanho;
};
```

Seja abaixo a definição das operações para esse **TAD Lista**:

```
Lista * lista_cria ();
void lista_libera (Lista * p);
int lista_insere_inicio (Lista * p, int elemento);
int lista_remove_posicao (Lista * p, int posicao);
int lista_obtem_elemento  (Lista * p, int posicao);
int lista_obtem_tamanho  (Lista * p);
```

Escreva um programa que cria uma lista linear, conforme as definições acima, cuja inserção de elementos sempre ocorre no início da lista. Contudo, se após cada inserção de um elemento, o novo tamanho da lista exceder ao tamanho máximo (o qual é fornecido pela entrada), então o elemento do meio da lista será imediatamente removido (para respeitar o tamanho máximo).
Muito importante: O programa (função **main**) usará somente as operações previstas para o TAD.
A seguir estão alguns casos para a inserção de elementos. O elemento inserido está em negrito.

| Lista após uma inserção | Tamanho máximo | Posição do elemento a remover | Conteúdo da lista |
|---|---|---|---|
| **11** 33 | 1 | 1 | 33 |
| **45** 12 77 | 2 | 2 | 45 77 |
| **17** 22 27 44 | 3 | 2 | 17 27 44 |
| **15** 12 31 67 49 | 4 | 3 | 15 12 67 49 |
| **22** 88 33 68 19 66 | 5 | 3 | 22 88 68 19 66 |
| **37** 12 30 82 27 72 31 | 6 | 4 | 37 12 30 27 72 31 |

Entrada: A primeira linha contém dois números inteiros: **n** (n>0) que indica a quantidade de números a inserir na lista; e **m** (m>0) que se refere ao tamanho máximo da lista. A segunda linha possui os elementos a inserir, separados por um espaço em branco.
Saída: O conteúdo da lista após as inserções; os elementos são separados por um espaço em branco.
Exemplo

| Entrada | Saída |
|---|---|
| 5 3<br>1 2 3 4 5 | 5 2 1 |

**Questão 06) TAD Palavra**
Considere abaixo as definições das estruturas de dados para um **TAD Palavra** que possui nome e seu significado (sinônimo), bem como a definição das operações para esse **TAD Palavra**:

```
typedef struct palavra Palavra;
struct palavra {
      char * nome;
      char * sinonimo;
};
```

```
Palavra * palavra_cria (char * nome, char * sinonimo);
void palavra_libera (Palavra * p);
char * palavra_obtem_nome (Palavra * p);
char * palavra_obtem_sinonimo (Palavra * p);
int palavra_compara (Palavra * p1, Palavra * p2);
```

Escreva um programa que cria um vetor de palavras (declaração **Palavra * vetor [n];** na função **main**), onde **n** é o tamanho do vetor, conforme as definições do TAD acima.
Muito importante: O programa (função **main**) usará somente as operações previstas para o TAD.

Entrada: A primeira linha contém um número **n** (n>0), que indica o número de palavras. As **n** linhas seguintes possui um **nome** e seu **sinônimo**, separados por um espaço (é garantido que nome e sinônimo não possuem espaços). Cada das demais linhas possui uma das seguintes operações:

- **C nome**, consulta **nome** no vetor de palavras;
- **F**, para finalizar o programa.

Saída: uma linha em resposta a cada operação do tipo **C**, para apresentar seu sinônimo ou INEXISTENTE, conforme exemplo abaixo. O conteúdo do vetor de palavras ordenado ao final, com cada nome e sinônimo em linha separada.
Exemplo

| Entrada | Saída |
|---|---|
| 5<br>nome7 sinonimo7<br>nome3 sinonimo3<br>nome1 sinonimo1<br>nome9 sinonimo9<br>nome2 sinonimo2<br>C nome3<br>C nome6<br>F | CONSULTA nome3 sinonimo3<br>CONSULTA nome6 INEXISTENTE<br>nome1 sinonimo1<br>nome2 sinonimo2<br>nome3 sinonimo3<br>nome7 sinonimo7<br>nome9 sinonimo9 |