

Due date: segunda-feira, 13 abril 2026, 23:59



 [Description](#)

 [Submission](#)

 [Edit](#)

 [Submission view](#)

 **Due date:** segunda-feira, 13 abril 2026, 23:59

 **Requested files:** main-sq.cpp, stack_queue.cpp, stack_queue.h ( [Download](#))

Type of work:  Individual work

Estruturas de Dados e Algoritmos Fundamentais
(ano letivo 2025-26)

Este e-fólio deve ser submetido em dois locais distintos:

- A resolução do programa é realizada neste recurso VIRTUAL PROGRAMMING LAB e-fólio A
- A resolução deve ainda ser acompanhada por um relatório constituído por um único ficheiro em formato pdf, entregue única e exclusivamente através do recurso "E-Fólio A", da turma respetiva.

ENUNCIADO

Pretende-se desenvolver um programa em linguagem C++11 padrão que aceite comandos para a gestão de duas estruturas de dados sequenciais de inteiros:

- **uma pilha (LIFO);**

- **uma fila (FIFO);**

ambas implementadas manualmente com recurso a listas simplesmente ligadas. Os itens armazenados são números inteiros (positivos ou negativos), representando simultaneamente o papel de chave e de informação.

As estruturas são inicializadas por defeito como vazias e são alteradas com comandos especificados num ficheiro de entrada fornecido ao programa pela entrada padrão (stdin em C e cin em C++), um comando por linha, podendo um comando ter vários argumentos, com o seguinte formato,

cmd arg0 arg1 ...

onde "cmd" indica o nome do comando a executar, "arg" é um tipo de argumento do comando e "..." a seguir a um argumento indica que este pode ser repetido várias vezes indefinidamente. Na descrição dos comandos, um argumento pode ser representado por: "item" indicando um inteiro; "qtd" indicando uma quantidade inteira maior ou igual a zero.

A lista dos comandos a implementar é a indicada a seguir. No caso de mensagens de saída de dados, o seu formato é indicado em estilo da linguagem C.

Operações sobre a pilha

stack_push item ...

Comando que insere itens na pilha pela ordem apresentada. O último item indicado deve ficar no topo da pilha.

stack_pop

Comando que remove o item do topo da pilha.

stack_top

Comando que imprime o item do topo da pilha. Formato "Pilha(top)= %d\n".

stack_print

Comando que imprime toda a pilha, do topo para a base. Formato "Pilha= %d %d ... \n".

stack_dim

Comando que imprime o número total de itens na pilha. Formato "Pilha tem %d itens\n".

stack_clear

Comando que remove todos os nós da pilha.

Operações sobre a fila

queue_enqueue item ...

Comando que insere itens no fim da fila pela ordem apresentada.

queue_dequeue

Comando que remove o item da frente da fila.

queue_front

Comando que imprime o item da frente da fila. Formato "Fila(front)= %d\n".

queue_back

Comando que imprime o item do fim da fila. Formato "Fila(back)= %d\n".

queue_print

Comando que imprime toda a fila, da frente para o fim. Formato "Fila= %d %d ... \n".

queue_dim

Comando que imprime o número total de itens na fila. Formato "Fila tem %d itens\n".

queue_clear

Comando que remove todos os nós da fila.

Comandos específicos

transfer_stack_queue qtd

Comando que move qtd itens da pilha para a fila, retirando sempre do topo da pilha e inserindo no fim da fila, um a um.

Exemplo: se a pilha contiver, do topo para a base, "7 5 2" e a fila contiver, da frente para o fim, "10 11", então após o comando

```
transfer_stack_queue 2
```

a pilha passa a conter "2" e a fila passa a conter "10 11 7 5".

reverse_queue

Comando que inverte a ordem dos itens atualmente existentes na fila. A estratégia a seguir é recorrer obrigatoriamente a uma pilha auxiliar implementada pelo próprio estudante, removendo os itens da fila e reinsertando-os após passarem pela pilha auxiliar.

Todos os comandos da pilha que não possam ser executados por a pilha estar vazia devem imprimir uma mensagem com o **formato** "Comando %s: Pilha vazia!\n".

Todos os comandos da fila que não possam ser executados por a fila estar vazia devem imprimir uma mensagem com o **formato** "Comando %s: Fila vazia!\n".

Isto inclui os comandos stack_clear e queue_clear quando a respectiva estrutura se encontrar vazia.

O comando transfer_stack_queue deve imprimir uma mensagem com o **formato** "Comando %s: Quantidade invalida!\n" sempre que qtd seja inválida ou superior ao número de itens atualmente existentes na pilha.

É da responsabilidade do programa assegurar que nenhuma operação que comprometa a estabilidade do programa é executada, por exemplo remover um item de uma pilha vazia ou aceder ao fim de uma fila vazia.

Projete as estruturas de dados (classes) adequadas ao programa que se pretende desenvolver. No que apenas respeita aos atributos (variáveis membro), o nó deve possuir o item armazenado e o apontador para o nó seguinte. A pilha deve possuir um apontador para o topo e um contador que a cada momento indica quantos nós tem a pilha. A fila deve possuir apontadores para o primeiro e último nós e um contador que a cada momento indica quantos nós tem a fila. Todos os nós e estruturas utilizadas no programa devem ser instâncias de classes (objetos) e não um conjunto de variáveis soltas ou isoladas.

No desenvolvimento do programa não é permitido utilizar algoritmos com nós falsos (dummy nodes).

Os métodos (funções membro) são livres. Deverá, na elaboração do programa, definir e criar os métodos e construtores que considerar mais adequados. O código dos métodos deve ser definido fora das classes, que apenas devem ter os respetivos protótipos. Os métodos e os comandos devem ser implementados também tendo em conta a sua eficiência.

O ficheiro de entrada (fornecido automaticamente pela entrada padrão) pode ter linhas em branco e o número de espaços que separa o comando e os argumentos entre si pode ser qualquer. Também podem existir linhas de comentário, caso em que começam obrigatoriamente pelo carácter '#' na 1ª posição.

Projete e teste uma versão do programa que implemente as especificações e comandos pedidos. O programa deve ler e processar o ficheiro de entrada uma linha inteira de cada vez, num ciclo ler-linha processar-linha até chegar ao fim do ficheiro de entrada. É estritamente proibido ler o ficheiro de entrada todo de uma vez para memória.

É importante saber ler o ficheiro de entrada corretamente e eficientemente sem usar código complexo. Sugere-se que seja lida uma linha de cada vez para uma string e analisado o primeiro carácter para ver se é um '#', caso em que a linha é um comentário. Caso não seja comentário, criar uma stringstream e usar o operador extrator >> para obter o nome do comando. Terá sucesso se não for uma linha em branco. Após saber o nome do comando, sabe-se quais os tipos de argumentos do comando que se seguem. Note-se que o operador extrator ignora ou salta caracteres brancos (espaços, tabs e new lines) antes de efetuar uma leitura. Programas que processem uma linha de entrada carácter a carácter serão penalizados devido à sua complexidade.

No desenvolvimento do programa em C++11 não devem ser utilizadas variáveis globais nem ser utilizada a STL, nomeadamente no que respeita às estruturas de dados e algoritmos estudados, devendo o aluno escrever o próprio código. Restrições aplicam-se aos includes <vector> <array> <deque> <forward_list> <list> <map> <queue> <set> <stack> <unordered_map> <unordered_set> e em parte de <algorithm>. Em caso de dúvida questionar o seu uso. Não existem restrições para <string>. Também não devem ser usados smart pointers.

Exemplo de dados entrada / saída padrão (Input: e Output: não fazem parte dos dados)

Input:

```
# Esta linha é um exemplo de comentário
stack_push 4 7 -2
stack_print
queue_enqueue 10 20
transfer_stack_queue 2
queue_print
stack_print
```

Output:

```
Pilha= -2 7 4
Fila= 10 20 -2 7
Pilha= 4
```

Atenção:

- Todas as linhas de saída são terminadas por \n sem espaços adicionais. No painel "Comments" podem não ser visíveis espaços adicionais da saída do programa, o que pode levar a pensar que o output está correto.
- No caso de desenvolver código em ambiente externo ao VPL (ex. Windows), para compatibilidade utilize codificação UTF-8 no editor de texto.

Critérios de correção:

- O programa desenvolvido difere significativamente das especificações e instruções do enunciado => 0 valores.
- O programa utiliza variáveis globais ou componentes da STL não permitidas no enunciado do VPL => 0 valores.
- O programa altera o nome das variáveis e classes definidas nas templates de código do VPL => 0 valores.
- O programa não compila ou produz avisos com g++ -Wall -std=c++11 => 0 valores.
- O código do programa não está correta e uniformemente indentado de modo a permitir a sua leitura fácil => 0 valores.
- O programa não está comentado => 0 valores. Os comentários no programa devem elucidar questões relevantes do código locais ao comentário e não o funcionamento geral do programa, que deve ser explicado no relatório.
- O programa é avaliado tendo como ponto de partida a percentagem de testes com resultado positivo. O nível

de simplicidade e qualidade do código também é avaliado. Programas considerados mal estruturados, demasiado complexos, confusos ou ineficientes podem ser penalizados até 50%.

- Apenas são considerados os resultados e o código dos programas constantes no recurso VPL.
- O e-fólio só é considerado entregue com a submissão do relatório em formato pdf no respetivo recurso no espaço turma. O formato do relatório é livre, com um máximo de 1 página útil além da capa/cabeçalho, com letra mínima 11. Este deve explicar sucintamente o funcionamento do programa a um nível geral, referindo os pontos principais e opções tomadas do programa que desenvolveu. Deve ainda indicar, de forma sucinta, a complexidade assintótica das operações `stack_push`, `stack_pop`, `queue_enqueue`, `queue_dequeue`, `transfer_stack_queue` e `reverse_queue`.
- Trabalhos com plágio comprovado ou muito similares entre si, independentemente da origem, serão anulados.

Nota ética: Nunca é de mais referir que o código a apresentar como solução para este e-fólio deve ser 100% original do aluno. A probabilidade de duas pessoas que efetivamente não comunicaram entre si apresentarem programas “quase iguais” é considerada nula. Isto é válido para qualquer par de alunos (cópia), assim como entre um aluno e qualquer outra pessoa, em particular através da Internet (cópia/plágio), onde existem inúmeras soluções e código para os mais variados problemas, em sites, fóruns, blogs, IA, etc.

Bom trabalho!

FIM

Requested files

main-sq.cpp

```
1  /*
2  ** file: main-sq.cpp
3  **
4  ** stack + queue with singly linked lists
5  ** UC: 21046 - EDAF @ UAb
6  ** e-fólio A 2025-26
7  **
8  ** Aluno: NNNNN - Nome Apelido
9  */
10
11 // Defina neste ficheiro:
12 //   A entrada/saída de dados
13 //   As instâncias das classes das estruturas de dados
14 //   A implementação dos comandos através dos métodos das classes
15 //   Código auxiliar
16 //   Não utilize variáveis globais!
17
18 #include <iostream>
19 #include <sstream>
20 #include <string>
21 #include "stack_queue.h"
22 using namespace std;
23
24 int main()
25 {
26     IStack stack;
27     IQueue queue;
28
29     string line;
30     while (getline(cin, line)) {
31         // TODO: ignorar linhas em branco e comentários
32         // TODO: processar cada comando com stringstream
33         (void)stack;
34         (void)queue;
35         (void)line;
36     }
37
38     return 0;
39 }
40
41 // EOF
42
```

stack_queue.cpp

```

1  /*
2  ** file: stack_queue.cpp
3  **
4  ** stack + queue with singly linked lists
5  ** UC: 21046 - EDAF @ UAb
6  ** e-fólio A 2025-26
7  **
8  ** Aluno: NNNNN - Nome Apelido
9  */
10
11 // Defina:
12 //   Em stack_queue.h as classes da estrutura de dados
13 //   Em stack_queue.cpp a implementação dos métodos das classes da estrutura de dados
14 // ** Não altere o nome das classes nem dos atributos obrigatórios! **
15
16 #include <iostream>
17 #include "stack_queue.h"
18 using namespace std;
19
20 INode::INode(int value, INode *pNext)
21 {
22     item = value;
23     next = pnext;
24 }
25
26 IStack::IStack()
27 {
28     n = 0;
29     ptop = 0;
30 }
31
32 IStack::~IStack()
33 {
34     clear();
35 }
36
37 bool IStack::empty() const
38 {
39     return n == 0;
40 }
41
42 int IStack::size() const
43 {
44     return n;
45 }
46
47 void IStack::push(int item)
48 {
49     // TODO
50     (void)item;
51 }
52
53 void IStack::pop()
54 {
55     // TODO
56 }
57
58 int IStack::top() const
59 {
60     // TODO
61     return 0;
62 }
63
64 void IStack::clear()
65 {
66     while (!empty()) {
67         pop();
68     }
69 }
70
71 void IStack::print() const
72 {
73     // TODO
74 }
75
76 IQueue::IQueue()
77 {
78     n = 0;
79     pfirst = 0;
80     plast = 0;
81 }
82
83 IQueue::~IQueue()
84 {
85     clear();
86 }
87
88 bool IQueue::empty() const
89 {
90     return n == 0;
91 }
92

```

```
91 }
92
93 int IQueue::size() const
94 {
95     return n;
96 }
97
98 void IQueue::enqueue(int item)
99 {
100     // TODO
101     (void)item;
102 }
103
104 void IQueue::dequeue()
105 {
106     // TODO
107 }
108
109 int IQueue::front() const
110 {
111     // TODO
112     return 0;
113 }
114
115 int IQueue::back() const
116 {
117     // TODO
118     return 0;
119 }
120
121 void IQueue::clear()
122 {
123     while (!empty()) {
124         dequeue();
125     }
126 }
127
128 void IQueue::print() const
129 {
130     // TODO
131 }
132
133 // EOF
134
```

stack_queue.h

```

1  /*
2  ** file: stack_queue.h
3  **
4  ** stack + queue with singly linked lists
5  ** UC: 21046 - EDAF @ UAb
6  ** e-fólio A 2025-26
7  **
8  ** Aluno: NNNNN - Nome Apelido
9  */
10
11 // Defina:
12 //   Em stack_queue.h as classes da estrutura de dados
13 //   Em stack_queue.cpp a implementação dos métodos das classes da estrutura de dados
14 // ** Não altere o nome das classes nem dos atributos obrigatórios! **
15
16 #ifndef _STACK_QUEUE_H
17 #define _STACK_QUEUE_H
18
19 struct INode {
20     // atributos obrigatórios
21     int item;           // informação em cada nó
22     INode *next;       // apontador para o nó seguinte
23
24     // outros atributos e métodos (protótipos) livres
25     INode(int value = 0, INode *pnext = 0);
26 };
27
28 class IStack {
29 private:
30     // atributos obrigatórios
31     int n;             // dimensão atual da pilha
32     INode *ptop;       // topo da pilha
33
34     // outros atributos e métodos (protótipos) livres
35 public:
36     IStack();
37     ~IStack();
38
39     bool empty() const;
40     int size() const;
41
42     void push(int item);
43     void pop();
44     int top() const;
45     void clear();
46
47     void print() const;
48 };
49
50 class IQueue {
51 private:
52     // atributos obrigatórios
53     int n;             // dimensão atual da fila
54     INode *pfirst;     // primeiro nó da fila
55     INode *plast;      // último nó da fila
56
57     // outros atributos e métodos (protótipos) livres
58 public:
59     IQueue();
60     ~IQueue();
61
62     bool empty() const;
63     int size() const;
64
65     void enqueue(int item);
66     void dequeue();
67     int front() const;
68     int back() const;
69     void clear();
70
71     void print() const;
72 };
73
74 #endif
75 // EOF
76

```