

Melhores momentos

AULA 14

Filas



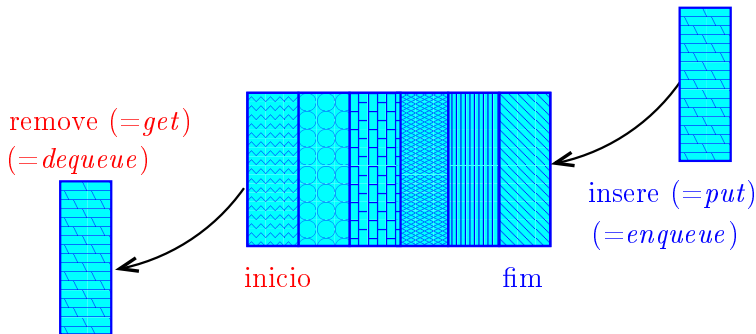
Fonte: <http://justoutsidetheboxcartoon.com/>

PF 5.1

<http://www.ime.usp.br/~pf/algoritmos/aulas/fila.html>

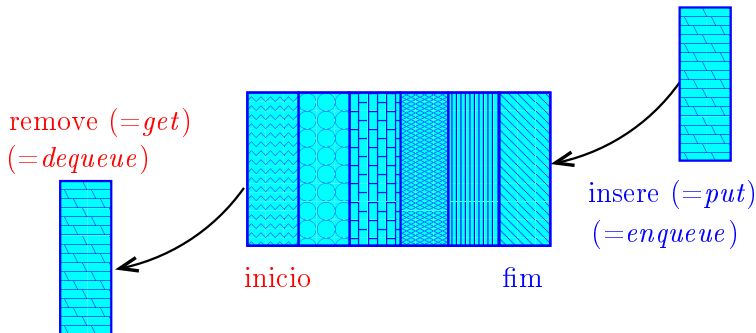
Filas

Uma **fila** (= *queue*) é uma lista dinâmica em que todas as **inserções** são feitas em uma extremidade chamada de **fim** e todas as **remoções** são feitas na outra extremidade chamada de **início**.



Filas

Assim, o primeiro objeto a ser removido de uma fila é o primeiro que foi inserido. Esta política de manipulação é conhecida pela sigla **FIFO** (= *First In First Out*)



Cliente de uma fila

A função `distancias` usa uma fila através de uma interface (`queueInit()`, `queuePut()` ...).
Não sabemos como a fila está implementada e essa informação **não interessa** ao `cliente`.

```
int *
distancias (int n, int **a, int c) {
    int *dist; /* dist[i] = dist. de c a
i*/
    int j;
```

distancias

```
queueInit(n); /* inicialize a fila */

/* aloque vetor de distancias */
dist = mallocSafe(n* sizeof(int));

/* inicialize o vetor de distancias */
for (j = 0; j < n; j++)
    dist[j] = n; /* distancia n = infinito
*/
dist[c] = 0;

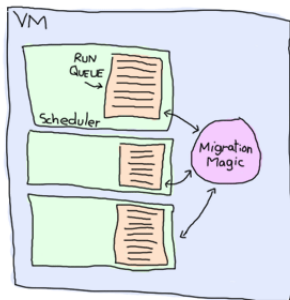
queuePut(c); /* coloque c na fila */
```

distancias

```
while (!queueEmpty()) {
    int i = queueGet();
    int di = dist[i];
    for (j = 0; j < n; j++)
        if (a[i][j] == 1 && dist[j] > di+1){
            dist[j] = di + 1;
            queuePut(j);
        }
}
queueFree();
return dist;
}
```

AULA 15

Implementações de filas



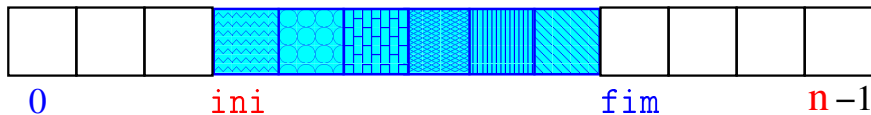
Fonte: <http://learnyousomeerlang.com/>

PF 5.3

<http://www.ime.usp.br/~pf/algoritmos/aulas/fila.html>

Filas em vetores

A fila será armazenada em um vetor $q[0 \dots n-1]$.



O índice `ini` indica o `primeiro` da fila.

O índice `fim-1` indica o `último` da fila.

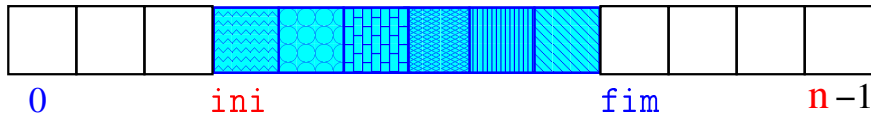
`fim` é a `primeira posição vaga` da fila.

A fila está `vazia` se "`ini == fim`".

A fila está `cheia` se "`fim == n`".

Filas em vetores

A fila será armazenada em um vetor $q[0 \dots n-1]$.



Para *remover* (= *dequeue*=*get*) um elemento faça

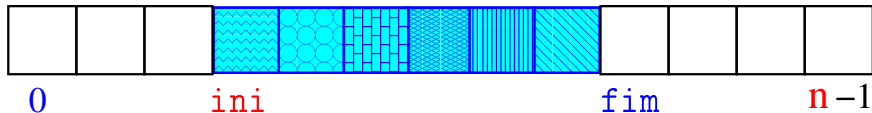
```
x = q[ini++];
```

que é equivalente a

```
x = q[ini];  
ini += 1;
```

Filas em vetores

A fila será armazenada em um vetor $q[0 \dots n-1]$.



Para *inserir* (= *queue* = *put*) um elemento faça

```
q[fim++] = x;
```

que é equivalente a

```
q[fim] = x;  
fim += 1;
```

Interface item.h

```
/*  
 * item.h  
 */  
typedef int Item;
```

Interface queue.h

```
/*
 * queue.h
 * INTERFACE: funcoes para manipular uma
 * fila
 */
void queueInit(int);
int queueEmpty();
void queuePut(Item);
Item queueGet();
void queueFree();
```

Implementação queue.c

```
#include <stdlib.h>
#include <stdio.h>
#include "item.h"
/*
 * FILA: implementacao em vetor
 */
static Item *q;
static int ini;
static int fim;
```

Implementação queue.c

```
void
queueInit(int n)
{
    q = mallocSafe(n * sizeof(Item));
    ini = fim = 0;
}

int
queueEmpty()
{
    return ini == fim;
}
```


Implementação queue.c

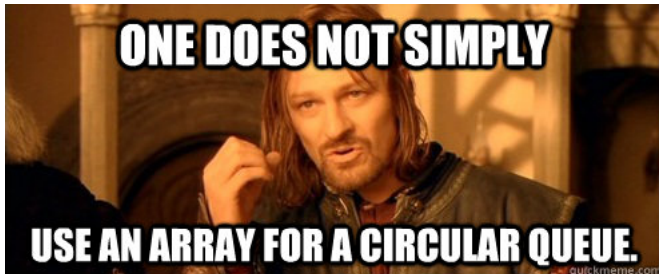
```
void  
queuePut(Item item)  
{  
    q[fim++] = item;  
}
```

```
Item  
queueGet()  
{  
    return q[ini++];  
}
```

Implementação queue.c

```
void  
queueFree()  
{  
    free(q);  
}
```

Filas em vetores circulares



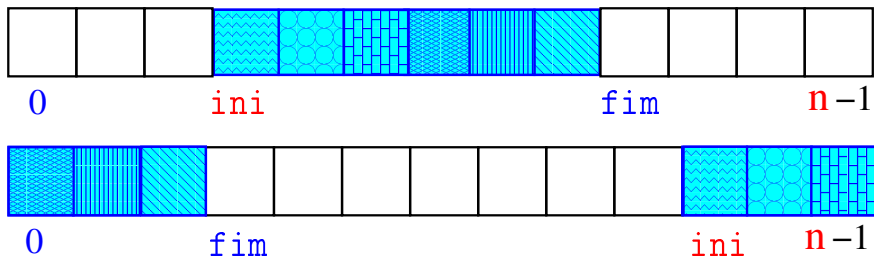
Fonte: <http://www.quickmeme.com/>

PF 5.1

<http://www.ime.usp.br/~pf/algoritmos/aulas/fila.html>

Fila implementada circularmente em um vetor

A fila será armazenada em um vetor $q[0 \dots n-1]$.



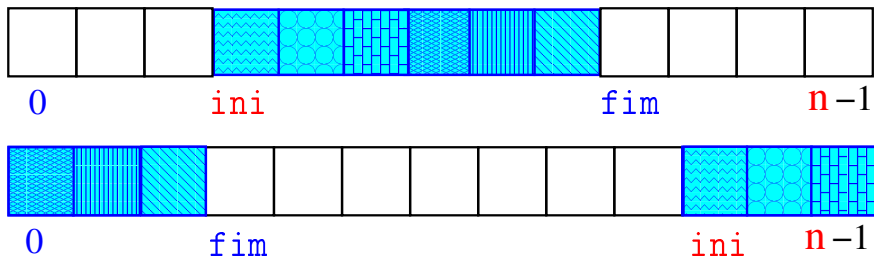
Temos que $0 \leq ini < n$ e $0 \leq fim < n$

Não supomos $ini \leq fim$

O índice ini indica o primeiro da fila.

Fila implementada circularmente em um vetor

A fila será armazenada em um vetor $q[0 \dots n-1]$.



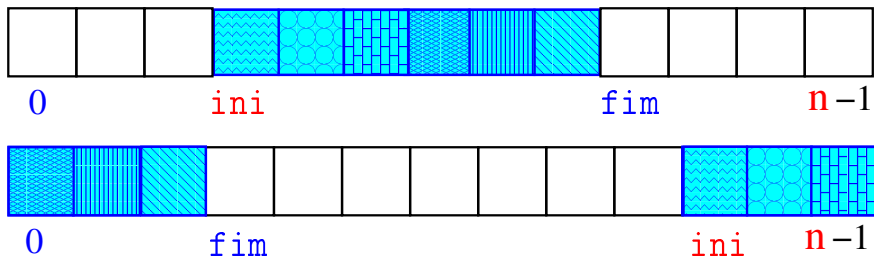
A fila está vazia se $ini == fim$

A fila está cheia se $fim+1 == ini$ ou

$fim+1 == n$ e $ini == 0$

Fila implementada circularesmente em um vetor

A fila será armazenada em um vetor $q[0 \dots n-1]$.

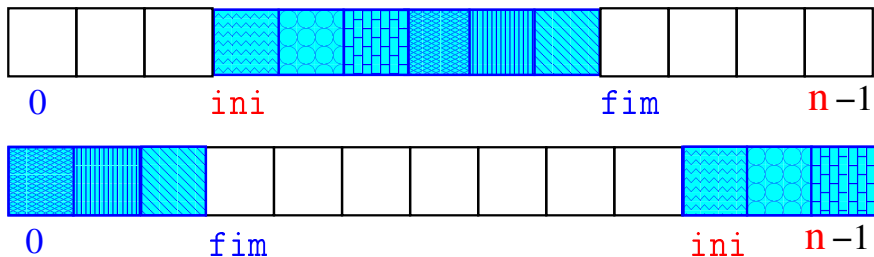


A fila está **vazia** se “ $ini == fim$ ”

A fila está **cheia** se “ $(fim+1) \% n == ini$ ”

Fila implementada circularmente em um vetor

A fila será armazenada em um vetor $q[0 \dots n-1]$.

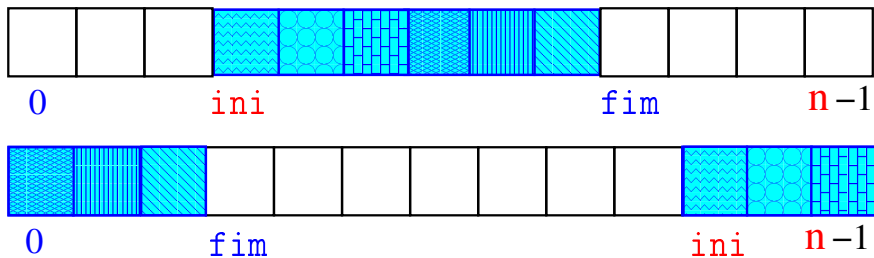


A posição fim **sempre está desocupada**

Isto é importante para distinguir fila **vazia** de **cheia**

Fila implementada circularmente em um vetor

A fila será armazenada em um vetor $q[0 \dots n-1]$.

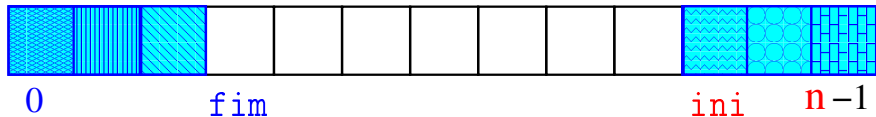


Para *remover* (= *dequeue* = *get*) um elemento faça

```
x = q[ini++];
```

```
if (ini == n) ini = 0;
```


Uma implementação circular em um vetor



Para *inserir* (= *queue* = *put*) um elemento faça

```
if((fim+1)%n == ini) {  
    printf("Fila cheia!\n");  
    exit(EXIT_FAILURE);  
}  
q[fim++] = x;  
if(fim == n) fim = 0;
```

Interface item.h

```
/*  
 * item.h  
 */  
typedef int Item;
```

Interface queue.h

```
/*  
 * queue.h  
 * INTERFACE: funcoes para manipular uma  
 * fila  
 */  
void queueInit(int);  
int queueEmpty();  
void queuePut(Item);  
Item queueGet();  
void queueFree();
```

Implementação queue.c

```
#include <stdlib.h>
#include <stdio.h>
#include "item.h"
/*
 * FILA: implementacao em vetor
 */
static Item *q;
static int n; /* tamanho do vetor */
static int ini;
static int fim;
```

Implementação queue.c

```
void
queueInit(int N)
{
    n = N + 1;
    q = mallocSafe(n * sizeof(Item));
    ini = fim = 0;
}

int
queueEmpty()
{
    return ini == fim;
}
```

Implementação queue.c

```
void
queuePut(Item item)
{
    if ((fim+1)%n == ini) {
        printf("Fila vai transbordar!\n");
        exit(EXIT_FAILURE);
    }
    q[fim++] = item;
    if (fim == n) fim = 0;
}
```

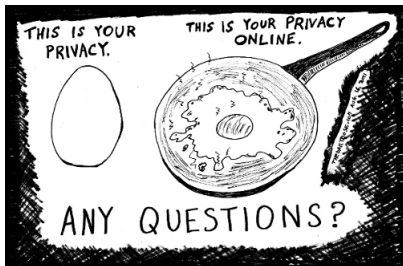
Implementação queue.c

Item

```
queueGet()  
{  
    int i = ini;  
    ini = (ini + 1) % n;  
    return q[i];  
}
```

```
void  
queueFree()  
{  
    free(q);  
}
```

Filas em listas encadeadas



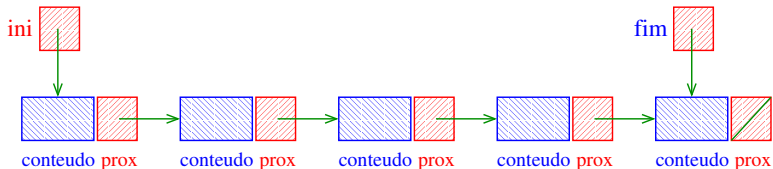
Fonte: <https://funnyjokesandlaughs.wordpress.com/>

PF 5.1

<http://www.ime.usp.br/~pf/algoritmos/aulas/fila.html>

Fila implementada em uma lista encadeada

A fila será armazenada em uma lista encadeada.



O ponteiro **ini** aponta para o **primeiro** da fila .

O ponteiro **fim** aponta para o **último** da fila .

ini->**conteudo** é o **primeiro elemento** da fila .

fim->**conteudo** é o **último elemento** da fila .

A fila está **vazia** se "**ini** == **NULL**".

Implementação queue.c

```
#include <stdlib.h>
#include <stdio.h>
#include "item.h"
/*
 * FILA: uma implementacao em lista
 * encadeada
 */
typedef struct queueNode* Link;
struct queueNode {
    Item conteudo;
    Link prox;
};
static Link ini, fim;
```

Implementação queue.c

```
static Link  
new(Item item, Link prox)  
{  
    Link p = mallocSafe(sizeof *p);  
    p->conteudo = item;  
    p->prox = prox;  
    return p;  
}
```

Implementação queue.c

```
void
queueInit(int n)
{
    ini = NULL;
}

int
queueEmpty()
{
    return ini == fim == NULL;
}
```

Implementação queue.c

```
void
queuePut(Item item)
{
    if (ini == NULL)
    {
        ini = fim = new(item, NULL);
        return;
    }
    fim->prox = new(item, NULL);
    fim = fim->prox;
}
```

Implementação queue.c

Item

queueGet()

```
{  
    Link p = ini;  
    Item item = ini->conteudo;  
  
    ini = ini->prox;  
    free(p);  
    return item;  
}
```

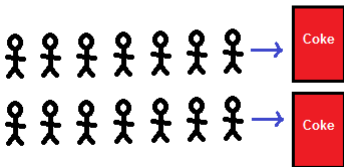
Implementação queue.c

```
void
queueFree()
{
    while (ini != NULL)
    {
        Link t = ini->prox;
        free(ini);
        ini = t;
    }
}
```

Filas em listas encadeadas com cabeça



Concurrent: 2 queues, 1 vending machine



Parallel: 2 queues, 2 vending machines

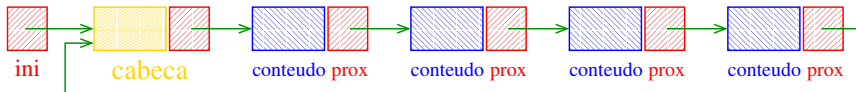
Fonte: <http://yosefk.com/>

PF 5.1

<http://www.ime.usp.br/~pf/algoritmos/aulas/fila.html>

Fila implementada em uma lista com cabeça

A fila será armazenada em uma **lista encadeada circular** com **cabeça**.



O ponteiro **ini** aponta para a **cabeça** da lista.

ini->**prox**->**conteudo** é **primeiro elemento** da fila.

A fila está **vazia** se "**ini**->**prox** == **ini**".

Implementação queue.c

```
#include <stdlib.h>
#include <stdio.h>
#include "item.h"
/*
 * FILA: uma implementacao em lista
 * encadeada circular com cabeca
 */
typedef struct queueNode* Link;
struct queueNode {
    Item conteudo;
    Link prox;
};
static Link ini;
```

Implementação queue.c

```
void
queueInit(int n)
{
    ini = mallocSafe(sizeof *ini);
    ini->prox = ini;
}
```

```
int
queueEmpty()
{
    return ini->prox == ini;
}
```

Implementação queue.c

```
void
queuePut(Item item)
{
    Link nova = mallocSafe(sizeof *nova);

    nova->prox = ini->prox;
    ini->prox = nova;

    /* insira item na celula cabeca (!) */
    ini->conteudo = item;

    /* mude a cabeca para nova (!) */
    ini = nova;
}
```

Implementação queue.c

Item

```
queueGet()
```

```
{
```

```
    Link p = ini->prox;
```

```
    Item item = p->conteudo;
```

```
    ini->prox = p->prox;
```

```
    free(p);
```

```
    return item;
```

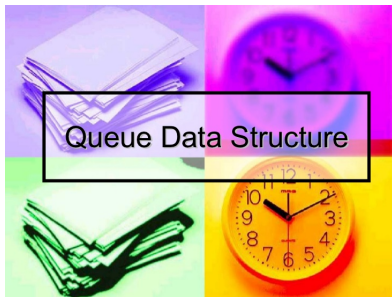
```
}
```

Implementação queue.c

```
void
queueFree()
{
    Link p = ini->prox;

    while (p != ini)
    {
        Link t = p->prox;
        free(p);
        p = t;
    }
    free(ini);
}
```

FilaS em listaS encadeadaS

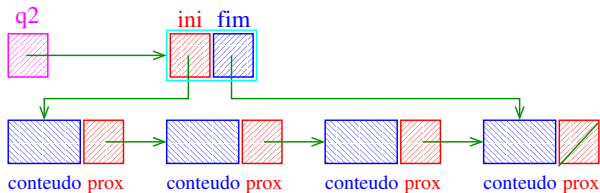
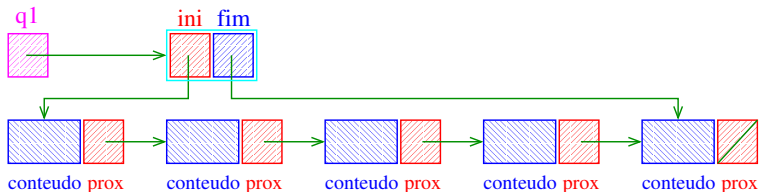


Fonte: <http://www.slideshare.net/>

S 4.8

FilaS implementadaS em listaS encadeadaS

As filas serão armazenada em listas encadeadas.



FilaS implementadaS em listaS encadeadaS

Uma fila `q` é um ponteiro para uma `struct` com campos `ini` e `fim`.

Para cada fila `q` há ponteiros `ini` e `fim`.

`q->ini->conteudo` é o primeiro elemento da fila `q`.

`q->fim->conteudo` é o último elemento da fila `q`.

A fila `q` está vazia se “`q->ini == NULL`”.

Interface item.h

```
/*  
 * item.h  
 */  
typedef int Item;
```

Interface queue.h

```
/*
 * queue.h
 * INTERFACE: funcoes para manipular filas
 * ATENCAO: Esta interface permite que
 * varias filas sejam utilizadas.
 */
typedef struct queue *Queue;

Queue queueInit(int);
int queueEmpty(Queue);
void queuePut(Queue, Item);
Item queueGet(Queue);
void queueFree(Queue);
```

distancias

A função `distancias` recebe um inteiro `n`, uma matriz `A` representando uma rede de estradas entre `n` cidades e uma cidade `c` e devolve um vetor `d` que registra a distância da cidade `c` a cada uma das outras: `d[i]` é a distância de `c` a `i`.

```
int *
distancias (int n, int **A, int c) {
    int *d; /* d[i] = distancia de c a i*/
    int j;
    Queue q;
```

distancias

```
/* aloque vetor de distancias */  
d = mallocSafe(n* sizeof(int));  
  
q = queueInit(n); /* crie uma fila */  
  
/* inicialize o vetor de distancias */  
for (j = 0; j < n; j++)  
    d[j] = n; /* distancia n = infinito */  
d[c] = 0;  
  
queuePut(q, c); /* coloque c na fila */
```

distancias

```
while (!queueEmpty(q)) {
    int i = queueGet(q);
    int di = d[i];
    for (j = 0; j < n; j++)
        if (A[i][j] == 1 && d[j] > di+1) {
            d[j] = di + 1;
            queuePut(q, j);
        }
}
queueFree(q);
return d;
}
```

Implementação queue.c

```
/*
 * FILA: uma implementacao em lista
 * encadeada
 */
typedef struct queueNode* Link;
struct queueNode {
    Item conteudo;
    Link prox;
};
struct queue {
    Link ini, fim;
};
typedef struct queue *Queue;
```

Implementação queue.c

```
static Link  
new(Item item, Link prox)  
{  
    Link p = mallocSafe(sizeof *p);  
    p->conteudo = item;  
    p->prox = prox;  
    return p;  
}
```


Implementação queue.c

Queue

```
queueInit(int n)
{
    Queue q = mallocSafe(sizeof *q);
    q->ini = NULL;
    return q;
}
```

```
int
queueEmpty(Queue q)
{
    return q->ini == NULL;
}
```

Implementação queue.c

```
void
queuePut(Queue q, Item item)
{
    if (q->ini == NULL)
    {
        q->ini = new(item, NULL);
        q->fim = q->ini;
        return;
    }
    q->fim->prox = new(item, NULL);
    q->fim = q->fim->prox;
}
```

Implementação queue.c

Item

```
queueGet(Queue q)
{
    Link p = q->ini;
    Item item = q->ini->conteudo;

    q->ini = q->ini->prox;
    free(p);
    return item;
}
```

Implementação queue.c

```
void
queueFree(Queue q)
{
    while (q->ini != NULL)
    {
        Link t = q->ini->prox;
        free(q->ini);
        q->ini = t;
    }
    free(q);
}
```

Compilação

cria o obj **queue.o**

```
> gcc -Wall -O2 -ansi -pedantic  
-Wno-unused-result -c queue.c
```

cria o obj **distancias.o**

```
> gcc -Wall -O2 -ansi -pedantic  
-Wno-unused-result \  
-c distancias.c
```

cria o executável **distancia**

```
> gcc queue.o distancia.o -o distancia
```

Makefile

Hmmm. Ler o tópico **Makefile** no fórum.

```
distancia: distancia.o queue.o
    gcc distancia.o queue.o -o distancia
```

```
distancia.o: distancia.c
    gcc -Wall -O2 -ansi -pedantic \
        -Wno-unused-result -c distancia.c
```

```
queue.o: queue.c item.h
    gcc -Wall -O2 -ansi -pedantic \
        -Wno-unused-result -c queue.c
```