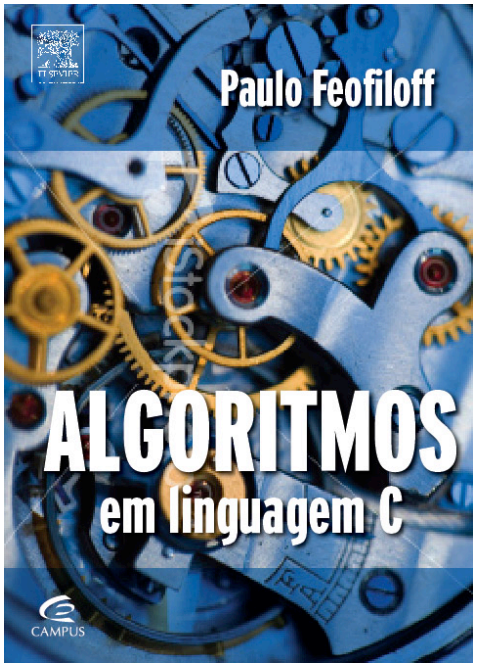
"Algoritmos em linguagem C"
Paulo Feofiloff
editora Campus/Elsevier, 2009




www.ime.usp.br/~pf/algoritmos-livro/

## Algoritmo de remoção

Remove o elemento de índice $k$ do vetor $v[0\,..\,n-1]$ e devolve o novo valor de $n$. Supõe $0 \le k < n$.

```
int Remove (int k, int v[], int n) {
   int j;
   for (j = k; j < n-1; j++)
      v[j] = v[j+1];
   return n - 1;
}
```

- funciona bem mesmo quando $k = n - 1$ ou $k = 0$
- exemplo de uso: `n = Remove (51, v, n);`

## Algoritmo de inserção

Insere $y$ entre as posições $k-1$ e $k$ do vetor $v[0\,..\,n-1]$ e devolve o novo valor de $n$. Supõe que $0 \le k \le n$.

```
int Insere (int k, int y, int v[], int n) {
   int j;
   for (j = n; j > k; j--)
      v[j] = v[j-1];
   v[k] = y;
   return n + 1;
}
```

- estamos supondo $n < \mathtt{N}$
- exemplo de uso: `n = Insere(51, 999, v, n);`

## Problema de busca-e-remoção

Remover todos os elementos nulos de um vetor $v[0\,..\,n-1]$.

## Algoritmo

Remove todos os elementos nulos de $v[0\,..\,n-1]$,
deixa o resultado em $v[0\,..\,i-1]$, e devolve o valor de $i$.

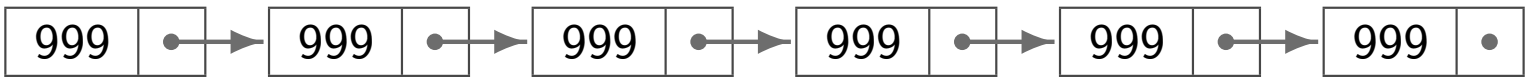
```
int RemoveZeros (int v[], int n) {
   int i = 0, j;
   for (j = 0; j < n; j++)
      if (v[j] != 0) {
         v[i] = v[j];
         i += 1;
      }
   return i;
}
```

## Ponteiros

- ponteiro é um tipo de variável capaz de armazenar endereços
- se p = &x então dizemos "p aponta para x"
- se p é um ponteiro então *p é o valor do objeto apontado por p

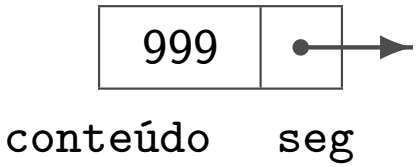

representação esquemática

# Listas encadeadas



## Estrutura de uma célula

```
struct cel {
   int          conteúdo;
   struct cel *seg;    /* seguinte */
};
```



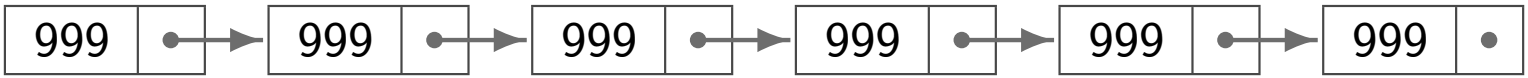
## Células são um novo tipo de dados

```
typedef struct cel célula;
```

## Definição de uma célula e de um ponteiro para célula

```
célula  c;
célula *p;
```

- conteúdo da célula: `c.conteúdo` / `p->conteúdo`
- endereço da célula seguinte: `c.seg` / `p->seg`




última célula da lista: `p->seg` vale `NULL`

## Exemplos: imprime lista com e sem cabeça

O algoritmo imprime o conteúdo de uma lista `lst` sem cabeça.

```
void Imprima (célula *lst) {
   célula *p;
   for (p = lst; p != NULL; p = p->seg)
      printf ("%d\n", p->conteúdo);
}
```

Imprime o conteúdo de uma lista `lst` com cabeça.

```
void Imprima (célula *lst) {
   célula *p;
   for (p = lst->seg; p != NULL; p = p->seg)
      printf ("%d\n", p->conteúdo);
}
```

## Algoritmo de busca

Recebe um inteiro $x$ e uma lista `lst` com cabeça.
Devolve o endereço de uma célula que contém $x$
ou devolve `NULL` se tal célula não existe.

```
célula *Busca (int x, célula *lst) {
   célula *p;
   p = lst->seg;
   while (p != NULL && p->conteúdo != x)
      p = p->seg;
   return p;
}
```

## Versão recursiva

```
célula *BuscaR (int x, célula *lst) {
   if (lst->seg == NULL)
      return NULL;
   if (lst->seg->conteúdo == x)
      return lst->seg;
   return BuscaR (x, lst->seg);
}
```

## Algoritmo de remoção de uma célula

Recebe o endereço `p` de uma célula em uma lista
e remove da lista a célula `p->seg`.
Supõe que `p` $\neq$ `NULL` e `p->seg` $\neq$ `NULL`.

```
void Remove (célula *p) {
    célula *lixo;
    lixo = p->seg;
    p->seg = lixo->seg;
    free (lixo);
}
```

## Algoritmo de inserção de nova célula

Insere uma nova célula em uma lista
entre a célula `p` e a seguinte (supõe `p` $\neq$ `NULL`).
A nova célula terá conteúdo $y$.

```
void Insere (int y, célula *p) {
    célula *nova;
    nova = malloc (sizeof (célula));
    nova->conteúdo = y;
    nova->seg = p->seg;
    p->seg = nova;
}
```

# Filas

# Fila implementada em vetor


uma fila `f[s..t-1]`

## Remove elemento da fila

```
x = f[s++];                    /* x = f[s]; s += 1; */
```

## Insere y na fila

```
f[t++] = y;                    /* f[t] = y; t += 1; */
```

# Fila implementada em vetor


uma fila `f[s..t-1]`

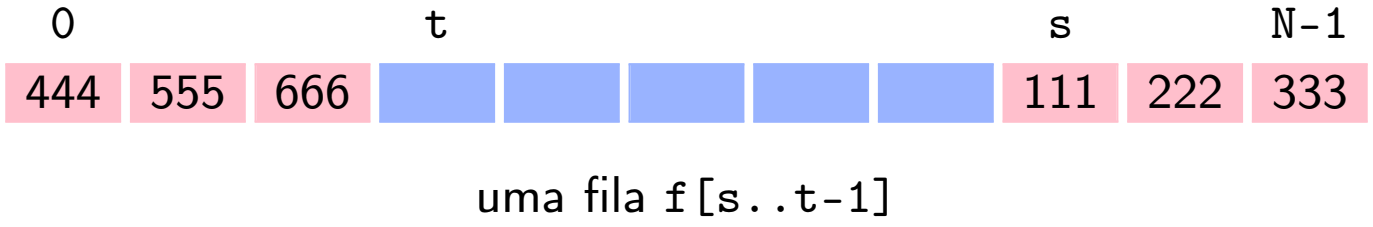
## Remove elemento da fila

```
x = f[s++];                    /* x = f[s]; s += 1; */
```

## Insere `y` na fila

```
f[t++] = y;                    /* f[t] = y; t += 1; */
```

# Implementação circular da fila



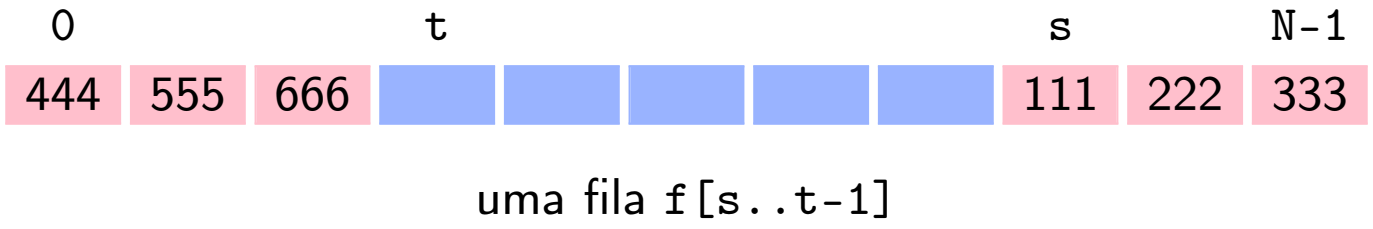

## Remove elemento da fila

```
x = f[s++];
if (s == N) s = 0;
```

## Insere y na fila

```
f[t++] = y;
if (t == N) t = 0;
```

# Implementação circular da fila




## Remove elemento da fila

```
x = f[s++];
if (s == N) s = 0;
```

## Insere y na fila

```
f[t++] = y;
if (t == N) t = 0;
```

# Fila implementada em lista encadeada

```
typedef struct cel {
   int          valor;
   struct cel *seg;
} célula;
```

## Decisões de projeto

- lista sem cabeça
- primeira célula: início da fila
- última célula: fim da fila

## Fila vazia

```
célula *s, *t;  /* s aponta primeiro elemento da fila */
s = t = NULL;   /* t aponta último elemento da fila */
```

# Fila implementada em lista encadeada

```
typedef struct cel {
   int          valor;
   struct cel *seg;
} célula;
```

## Decisões de projeto

- lista sem cabeça
- primeira célula: início da fila
- última célula: fim da fila

## Fila vazia

```
célula *s, *t;   /* s aponta primeiro elemento da fila */
s = t = NULL;    /* t aponta último elemento da fila */
```

## Remove elemento da fila

Recebe endereços `es` e `et` das variáveis `s` e `t` respectivamente.
Supõe que fila não está vazia e remove um elemento da fila.
Devolve o elemento removido.

```
int Remove (célula **es, célula **et) {
   célula *p;
   int x;
   p = *es;
   /* p aponta o primeiro elemento da fila */
   x = p->valor;
   *es = p->seg;
   free (p);
   if (*es == NULL) *et = NULL;
   return x;
}
```

## Insere elemento na fila

Recebe endereços `es` e `et` das variáveis `s` e `t` respectivamente.
Insere um novo elemento com valor `y` na fila.
Atualiza os valores de `s` e `t`.

```
void Insere (int y, célula **es, célula **et) {
   célula *nova;
   nova = malloc (sizeof (célula));
   nova->valor = y;
   nova->seg = NULL;
   if (*et == NULL) *et = *es = nova;
   else {
      (*et)->seg = nova;
      *et = nova;
   }
}
```

# Pilhas

# Pilha implementada em vetor


uma pilha `p[0..t-1]`

## Remove elemento da pilha

```
x = p[--t];                    /* t -= 1; x = p[t]; */
```

## Insere y na pilha

```
p[t++] = y;                    /* p[t] = y; t += 1; */
```

# Pilha implementada em vetor


uma pilha `p[0..t-1]`

## Remove elemento da pilha

```
x = p[--t];                    /* t -= 1; x = p[t]; */
```

## Insere `y` na pilha

```
p[t++] = y;                    /* p[t] = y; t += 1; */
```

# Aplicação: parênteses e chaves

- expressao bem-formada: `(){(){}}`
- expressao malformada: `({)}`

## Algoritmo

Devolve $1$ se a string `s` contém uma seqüência bem-formada e devolve $0$ em caso contrário.

```
int BemFormada (char s[]) {
    char *p; int t;
    int n, i;
    n = strlen (s);
    p = malloc (n * sizeof (char));
    processo iterativo
    free (p);
    return t == 0;
}
```

*processo iterativo*

```
t = 0;
for (i = 0; s[i] != '\0'; i++) {
   /* p[0..t-1] é uma pilha */
   switch (s[i]) {
      case ')': if (t != 0 && p[t-1] == '(') --t;
                else return 0;
                break;
      case '}': if (t != 0 && p[t-1] == '{') --t;
                else return 0;
                break;
      default: p[t++] = s[i];
   }
}
```

# Aplicação: notação posfixa

## Notação infixa versus posfixa

| infixa | posfixa |
|---|---|
| `( A + B * C )` | `A B C * +` |
| `( A * ( B + C ) / D - E )` | `A B C + * D / E -` |
| `( A + B * ( C - D * ( E - F ) - G * H ) - I * 3 )` | `A B C D E F - * - G H * - * + I 3 * -` |
| `( A + B * C / D * E - F )` | `A B C * D / E * + F -` |
| `( A * ( B + ( C * ( D + ( E * ( F + G ) ) ) ) ) )` | `A B C D E F G + * + * + *` |

# Pilha implementada em lista encadeada

```
typedef struct cel {
   int          valor;
   struct cel *seg;
} célula;
```

## Decisões de projeto

- lista com cabeça
- segunda célula: topo da pilha

## Pilha vazia

```
célula cabeça;
célula *p;
p = &cabeça;  /* p->seg é o topo da pilha */
p->seg = NULL;
```

## Insere

```
void Empilha (int y, célula *p) {
   célula *nova;
   nova = malloc (sizeof (célula));
   nova->valor = y;
   nova->seg = p->seg;
   p->seg = nova;
}
```

## Remove

```
int Desempilha (célula *p) {
   int x; célula *q;
   q = p->seg;
   x = q->valor;
   p->seg = q->seg;
   free (q);
   return x;
}
```