Listas ligadas/listas encadeadas

- Leitura recomendada:

`http://www.ime.usp.br/~pf/algoritmos/aulas/lista.html`

Processamento elementar de listas

Definição mais restritiva (tipo de lista possivelmente mais comum):

Definição 1. *Uma lista ligada é Λ (ponteiro* `NULL`*) ou um ponteiro para uma célula contendo um item e um ponteiro para uma lista ligada.*

- ▷ Fim de lista: última célula contém Λ
- ▷ Célula:

```
typedef struct node *link;
struct node { Item item; link next; };
```

Processamento elementar de listas

- Exemplo: varredura de um vetor `v[]`

```
for (i = 0; i < N; i++)
  visit(v[i]);
```

- Exemplo: varredura de uma lista `t`

```
for (s = t; s != NULL; s = s->next)
  visit(s->item);
```

Equivalente:

```
for (s = t; s; s = s->next)
  visit(s->item);
```

Processamento elementar de listas

▷ Criação (e impressão) de uma lista ligada de N números aleatórios entre 0 e 999:

```
link t = malloc(sizeof *t), x = t;
t->item = rand() % 1000;
for (i = 2; i <= N; i++) {
  x = (x->next = malloc(sizeof *x));
  x->item = rand() % 1000;
}
x->next = NULL;
<imprima lista t>
```

Processamento elementar de listas

▷ Exemplo: inversão de uma lista

```
/* prog3.10.c */
link reverse(link x)
  { link t, y = x, r = NULL;
    while (y != NULL)
      { t = y->next; y->next = r; r = y; y = t; }
    return r;
  }
```

Exercício 2. *Escreva um programa* `ex` *como segue: (a)* `ex` *recebe como argumento um inteiro* $N > 0$*; (b)* `ex` *gera uma lista com* N *inteiros aleatórios; (c)* `ex` *imprime esses inteiros varrendo a lista gerada; (d)* `ex` *então inverte a lista com uma chamada da função* `reverse()` *acima e, finalmente, (e)* `ex` *imprime os inteiros novamente, varrendo a lista invertida.*

O problema de Josephus

- Veja

  `http://mathworld.wolfram.com/JosephusProblem.html`

O problema de Josephus

```
/* prog3.9.c */
#include <stdio.h>
#include <stdlib.h>

typedef struct node *link;
struct node { int item; link next; };
```

O problema de Josephus

```
/* prog3.9.c */
[...]
int main(int argc, char *argv[])
  { int i, N = atoi(argv[1]), M = atoi(argv[2]);
    link t = malloc(sizeof *t), x = t;
    t->item = 1; t->next = t;
    for (i = 2; i <= N; i++) {
      x = (x->next = malloc(sizeof *x));
      x->item = i; x->next = t;
    }
```

O problema de Josephus

```
/* prog3.9.c */
[...]
    while (x != x->next) {
      for (i = 1; i < M; i++) x = x->next;
      x->next = x->next->next;
    }
    printf("%d\n", x->item);
    return 0;
  }
```

▷ O que acontece com as células eliminadas da lista?

O problema de Josephus, bis

```
/* prog3.9b.c */
[...]
    while (x != x->next) {
      for (i = 1; i < M; i++) x = x->next;
      t = x->next;
      x->next = x->next->next;
      free(t);
    }
    printf("%d\n", x->item);
    return 0;
  }
```

Exercício

Exercício 3. *Suponha que temos uma lista com elemento inicial apontada por* `head` *e fim de lista indicada por* `NULL`*. O que está errado com o seguinte código para liberar esta lista?*

```
for (p = head; p != NULL; p = p->next)
  free(p);
```

▷ Compare:

```
for (p = head; p != NULL; p = q) {
  q = p->next;
  free(p);
}
```

Ordenação de uma lista: ordenação por inseção

```
/* prog3.11.c */
[...]
struct node heada, headb;
link t, u, x, a = &heada, b;

for (i = 0, t = a; i < N; i++) {
    t->next = malloc(sizeof *t);
    t = t->next; t->next = NULL;
    t->item = rand() % 1000;
  }
<imprima lista a->next>
```

Ordenação de uma lista: ordenação por inseção

```
(cont.)
b = &headb; b->next = NULL;
for (t = a->next; t != NULL; t = u) {
    u = t->next;
    for (x = b; x->next != NULL; x = x->next)
      if (x->next->item > t->item) break;
    t->next = x->next; x->next = t;
  }
<imprima lista b->next>
[...]
```

Exercício

Exercício 4. *Reescreva o Programa 3.11 sem as cabeças de lista* `heada` *e* `headb`*.*

▷ Observe que o código fica menos elegante!

Listas com cabeça/cauda

- Sem cabeça, sem cauda [“mais comum”]
- Com cabeça, sem cauda [exemplo da ordenação]
- Com cabeça e cauda

Com cabeça e cauda

- Inicialização:

```
link head = malloc(sizeof *head);
link z = malloc(sizeof *z);
head->next = z; z->next = z;
```

- Inserção de `t` após `x`:

```
t->next = x->next; x->next = t;
```

Com cabeça e cauda

- Remoção do elemento que segue `x`:

```
x->next = x->next->next; /* free()? */
```

- Varredura:

```
for (t = head->next; t != z; t = t->next)
```

- Vazio?

```
if (head->next == z)
```

Exercícios

Exercício 5. *Escreva um código que elimina da lista* `t` *as células nas posições pares da lista (2a. célula, 4a. célula, 6a. célula, etc). Você deve chamar* `free()` *para liberar a memória correspondente. Faça isso para listas seguindo várias convenções de cabeça e cauda (sem/com cabeça, sem/com cauda).*

Exercícios

Exercício 6. *Escreva uma função de protótipo*

```
void split(link t, link *even, link *odd);
```

que recebe uma lista `t` *e devolve duas listas (obtidas rearranjando-se as células da lista dada): uma com as células nas posições pares na lista original e uma outra com as células nas posições ímpares na lista original. Faça isso para listas seguindo várias convenções de cabeça e cauda (sem/com cabeça, sem/com cauda).*