EPs 1 e 2

- EP2: veja
  - `http://www.ime.usp.br/~fabricio/ep2/EP2.pdf`
- EP1: veja
  - `http://www.ime.usp.br/~yoshi/2006ii/mac122a/EPs/EP1/wc.pdf`
  - `http://www.ime.usp.br/~pf/CWEB/`
  - `http://www.ime.usp.br/~pf/algoritmos/apend/util.html`

Exercício

▷ Vimos ordenação de listas por inserção.

Exercício 1. *Implemente o algoritmo de ordenação por seleção para listas ligadas:*

```
/* dada uma lista a, cria lista ordenada b */
b = lista vazia
enquanto a está não-vazia
  encontre uma célula x com item máximo
  insira x no começo de b
```

Exercício 2. *Compare a eficiência da ordenação por seleção com a eficiência da ordenação por inserção.*

▷ Veremos algoritmos muito mais eficientes.

Listas ligadas/listas encadeadas

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

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

typedef link Node;
void initNodes(int);
link newNode(Item);
void freeNode(link);
void insertNext(link, link);
link deleteNext(link);
link Next(link);
Item theItem(link);
```

Josephus, novamente

```
/* prog3.13.c */
#include <stdio.h>
#include <stdlib.h>
#include "list.h"
int main(int argc, char *argv[])
  { int N = atoi(argv[1]), M = atoi(argv[2]);
    Item i;
    Node t, x;
    initNodes(N);
    for (i = 2, x = newNode(1); i <= N; i++)
      { t = newNode(i); insertNext(x, t); x = t; }
```

Josephus, novamente

```
  [...]
  while (x != Next(x))
    {
      for (i = 1; i < M; i++) x = Next(x);
      freeNode(deleteNext(x));
    }
  printf("%d\n", theItem(x));
  return 0;
}
```

Implementação das funções de manipulação de listas

```
/* prog3.14.c */
#include <stdlib.h>
#include "list.h"

link freelist;

void initNodes(int N)
  { int i;
    freelist = malloc((N+1)*(sizeof *freelist));
    for (i = 0; i < N+1; i++)
      freelist[i].next = &freelist[i+1];
    freelist[N].next = NULL;
  }
```

Implementação das funções de manipulação de listas

```
/* prog3.14.c */
(cont.)
link newNode(Item i)
  { link x = deleteNext(freelist);
    x->item = i; x->next = x;
    return x;
  }

void freeNode(link x)
  { insertNext(freelist, x); }

void insertNext(link x, link t)
  { t->next = x->next; x->next = t; }
```

Implementação das funções de manipulação de listas

```
/* prog3.14.c */
(cont.)
link deleteNext(link x)
  { link t = x->next; x->next = t->next; return t; }

link Next(link x)
  { return x->next; }

Item theItem(link x)
  { return x->item; }
```

Strings

- *string* $\equiv$ cadeia de caracteres terminada por `'\0'`
- `char *s;`
- biblioteca `string`:
  - `strlen()`, `strcpy()`, `strcmp()`, `strcat()`, ...
  - Veja
    - `http://www.ime.usp.br/~pf/algoritmos/aulas/strings.html`

Busca de padrão

```
/* prog3.15.c */
#include <stdio.h>
#include <stdlib.h>
#define N 10000
int main(int argc, char *argv[])
  { int i, j, t;
    char a[N], *p = argv[1];
    for (i = 0; i < N-1; a[i] = t, i++)
      if ((t = getchar()) == EOF) break;
    a[i] = 0;
```

Busca de padrão

```
/* prog3.15.c */
    [...]
    for (i = 0; a[i] != 0; i++)
      {
        for (j = 0; p[j] != 0; j++)
          if (a[i+j] != p[j]) break;
        if (p[j] == 0) printf("%d ", i);
      }
    printf("\n");
    return 0;
  }
```

Manipulação de strings: implementações elementares [vetores]

```
strlen(a):
  for (i = 0; a[i] != 0; i++) ; return i;

strcpy(a, b):
  for (i = 0; (a[i] = b[i]) != 0; i++) ;

strcmp(a, b):
  for (i = 0; a[i] == b[i]; i++)
    if (a[i] == 0) return 0;
  return a[i] - b[i];
```

Manipulação de strings: implementações elementares [ponteiros]

```
strlen(a):
  b = a; while (*b) b++; return b - a;

strcpy(a, b):
  while (*a++ = *b++) ;

strcmp(a, b):
  while (*a == *b) {
    if (*a == '\0') return 0;
    a++; b++;
    }
  return *a - *b;
```

Exercícios

Exercício 3. *Escreva um programa que recebe um string como argumento e lê uma lista de palavras e tem como saída as palavras que ocorrem no string.*

Exercício 4. *Escreva um programa que substitui segmentos de mais de um branco por um branco.*

Exercício 5. *Implemente o Programa 3.15 com ponteiros.*

Uma leitura de linha simples

▷ `fgets()`:

```
char *fgets(char *, int, FILE);
```

Exercícios

Exercício 6. *Escreva uma rotina de leitura de strings com o seguinte algoritmo. Seu algoritmo deve usa um 'buffer' alocado dinamicamente (buffer é simplemente um vetor de caracteres). Inicialmente, seu buffer tem um tamanho dado. A leitura dos caracteres de entrada é feita sequencialmente no buffer. Quando o buffer 'fica cheio' você deve alocar um novo buffer com o dobro do tamanho do buffer anterior, e copiar os caracteres já lidos para este novo buffer. A leitura sequencial dos caracteres deve continuar neste novo buffer.*

Uma solução

```
/* Arquivo getline.h */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*
 * INITIAL_BUFF_SIZE -- Tamanho inicial para o buffer usado em GetLine()
 */

#define INITIAL_BUFF_SIZE 20
char *GetLine();

extern char *prog_name;

#define out_of_mem(x) {  fprintf(stderr, "%s: Acabou memoria!\n", prog_name); \
                         exit(x); \
                      }
```

```
/* Arquivo getline.c */
#include "getline.h"

/*
 * GetLine(): le uma linha do stdin (terminada por '\n' ou fim de
 * arquivo) e devolve um ponteiro para um string contendo a linha
 * lida; a linha lida nao contem '\n'.  Memoria 'e alocada
 * dinamicamente para armazenar a linha lida.
 *
 * Obs.  Esta rotina foi copiada (com pequenas modificacoes) do livro
 * The Art and Science of C, por Eric Roberts.
 */

char *GetLine()
{
    char *line, *nline;
    int n = 0, ch, size;

    size = INITIAL_BUFF_SIZE;
    line = (char *) malloc(size + 1);
    if (!line)
        out_of_mem(-2);
```

```
[...]
while ((ch = getchar()) != '\n' && ch != EOF) {
    if (n == size) {
        size *= 2;
        nline = (char *) malloc(size + 1);
        if (!nline)
            out_of_mem(-3);
        strncpy(nline, line, n);
        free(line);
        line = nline;
    }
    line[n++] = ch;
}
if (n == 0 && ch == EOF) {
    free(line);
    return NULL;
}
line[n] = '\0';
```

```
    [...]
    nline = (char *) malloc(n + 1);
    if (!nline)
        out_of_mem(-4);
    strcpy(nline, line);
    free(line);
    return nline;
}
```