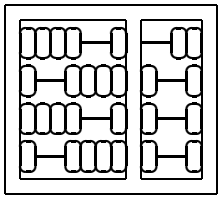

MC-102 Algoritmos e Programação de Computadores
IC-UNICAMP
Aula 29 - Exercícios Listas

1 Objetivos

Exercitar alocação dinâmica e listas encadeadas.

2 Motivação

Alocação dinâmica e listas são muito úteis quando não sabemos a quantidade de informação que o programa deverá suportar. Usando esses recursos podemos tornar os nossos programas mais robustos.

3 Exercícios

1. Escreva uma função que conta o número de nodos de uma lista encadeada. Faça uma versão recursiva e outra iterativa.

   Resposta:

```
int conta_nodos(Nodo *p)
{
  int cont=0;
  while(p!=NULL){
    p=p->prox;
    cont++;
  }
  return cont;
}

int conta_nodos_rec(Nodo *p)
{
  if(p==NULL)
    return 0;
  else
    return(1+conta_nodos_rec(p->prox));
}
```

2. Critique a função abaixo. Ao receber uma lista encadeada e um inteiro x, ela promete devolver o endereço de um nodo com contéudo x. Se tal nodo não existe, promete devolver NULL.

```
Nodo *busca (int x, Nodo *ini) {
   int achou;
```

```
Nodo *p;
achou = 0;
p = ini;
while (p != NULL && !achou) {
   if (p->info == x) achou = 1;
   p = p->prox; }
if (achou) return p;
else return NULL; }
```

Resposta: `A função retorna um ponteiro para o elemento após o x.`

3. Escreva uma função que faça um busca em uma lista ordenada. Faça versões recursiva e iterativa.

   Resposta:

```
Nodo * busca_ordenada(int x, Nodo *p)
{
  while ((p!=NULL)&&(x<p->info))
    p=p->prox;
  if((p!=NULL)&&(p->info==x))
    return p;
  else
    return NULL;
}

Nodo * buscaOrd_rec(int x, Nodo *p)
{
  if(p==NULL)
    return NULL;
  else
    if (p->info==x)
      return p;
    else
      if (p->info>x)
        return NULL;
      else
        return buscaOrd_rec(x,p->prox);
}
```

4. Por que a seguinte versão de insere não funciona?

```
void insere (int x, Nodo *p) {
   Nodo novo;
   novo.info = x;
   novo.prox = p->prox;
   p->prox = &novo; }
```

Resposta: `Porque assim que a função acaba o nodo criado é desalocado.`

5. Faça uma função para inserir um elemento no final da lista. Escreva uma versão recursiva e outra iterativa.

   Resposta:

```
void insere_fim (int x, Nodo **ini)
{
  Nodo *novo;
  Nodo *p;

  novo = (Nodo *)malloc (sizeof (Nodo));
  novo->info = x;
  novo->prox = NULL;
  if ((*ini)!=NULL){
    p=(*ini);
    while (p->prox != NULL)
      p=p->prox;
    p->prox=novo;
  }
  else
    (*ini)=novo;
}
```

```
void insere_rec (int x, Nodo **ini)
{
  Nodo *novo;
  if ((*ini)==NULL){
    novo = (Nodo *)malloc (sizeof (Nodo));
    novo->info = x;
    novo->prox = NULL;
    (*ini)=novo;
  } else
    insere_rec(x,(*ini)->prox);
}
```

6. Critique a seguinte versão da função remove:

```
void remove (Nodo *p) {
  Nodo *morta;
  morta = p->prox;
  if (morta->prox == NULL)
     p->prox = NULL;
  else
     p->prox = morta->prox;
  free (morta);
}
```

Resposta: `Não precisava do if e principalmente a memória não é desalocada com o comando free, o nodo fica perdido na memória.`

7. Escreva uma função para excluir o último elemento de uma lista. Faça versões iterativas e recursivas.

   Resposta:

```
void remove_fim (Nodo **ini)
{
  Nodo *apaga;
  Nodo *p;
  if ((*ini)!=NULL){
    if ((*ini)->prox!=NULL) {
      p=(*ini);
      while (p->prox->prox != NULL)
        p=p->prox;
      apaga=p->prox;
      p->prox=NULL;
    }
    else{
      apaga=(*ini);
      (*ini)=NULL;
    }
    free(apaga);
  }
}
```

```
void remove_rec (Nodo **ini)
{
  Nodo *novo;
  if ((*ini)==NULL)
    return;
  if((*ini)->prox==NULL){
    apaga=(*ini);
    (*ini)=NULL;
    free(apaga);
  }
  else
    remove_rec((*ini)->prox);
}
```

8. Escreva uma função para remover de uma lista encadeada todos os elementos que contêm y. Dica: faça ela recursiva.

   Resposta:

```
void removeX_rec (int x, Nodo **ini)
{
  Nodo *novo;
  if ((*ini)==NULL)
    return;
  if((*ini)->info==x){
    apaga=(*ini);
    (*ini)=(*ini)->prox;
    free(apaga);
  }
  else
    remove_rec(x,(*ini)->prox);
}
```