Estrutura de um programa

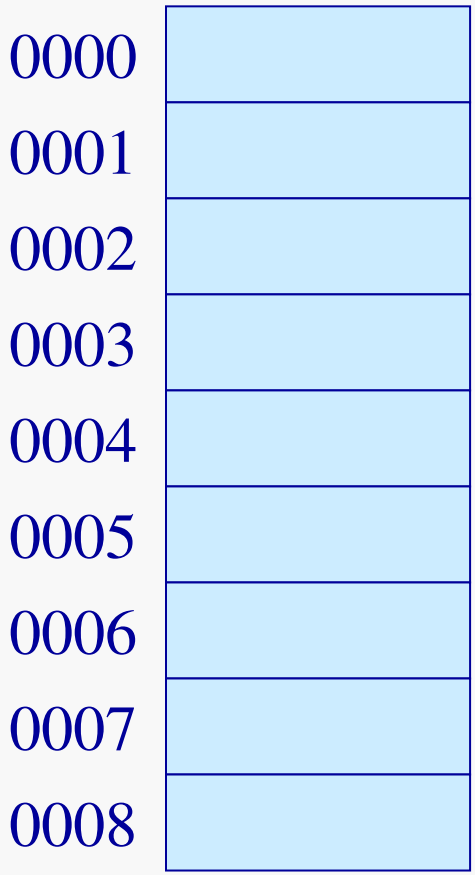
- Código Fonte – texto contendo o programa em C, Pascal, Java, etc.
- Código Executável – seqüência de instruções de máquina

Instruções

- Uma ‘instrução de máquina’ é formada por um código de operação e seus operandos
- Exemplos:
  - Soma, subtração, divisão, multiplicação
  - Comparação de valores
  - Desvios
- As instruções e seus operandos são mantidos na memória principal

Memória

- Conjunto de ‘posições’ de memória, todas de mesmo tamanho e acessadas pelas instruções através da sua ‘posição’ ou ‘endereço’
- Organização mais comum: memória formada por ‘posições’ de 8 bits (bytes).



Memória

- As posições da memória contém uma configuração qualquer de bits.
- O significado do conteúdo de uma área da memória é definido pelo uso que as instruções fazem do mesmo.
- Numa máquina cuja memória é organizada em bytes
  - Um caracter é representado num byte (8 bits)
  - Um número inteiro é representado, na maioria das vezes, por uma sequência de 4 bytes (32 bits)

Variáveis e endereços

- Uma variável, num programa em C,Pascal, Java, etc., é mantida em tempo de execução numa área de memória que contém o valor da variável.
- Os comandos do programa, são traduzidos para instruções, que operam com as variáveis referenciando-as através da sua posição ou *endereço* de memória.

Variáveis e endereços

- Um endereço é basicamente um numero inteiro que identifica uma posição da memória.
- O tipo apontador em C é na verdade um endereço de memória, ou seja um valor inteiro.

```
#include <stdio.h>
int main(){
  int a = 10;
  int *p = &a;
  *p = 20;
  printf("a:%d\n",a);
}
```

p contém o endereço de **a** portanto '**a**' e '***p***' se referem à mesma área de memória.

Parâmetros 'por referência'

```
#include <stdio.h>
void troca(int* a, int* b){
  int t;
  t=*a; *a = *b; *b = t;
}
int main() {
  int x = 10; int y = 20;
  ...
  troca(&a, &b);
  ...
}
```

A rigor, todos os parâmetros em C são passados por valor. O programa deve tratar os parâmetros passados 'por referência' de forma diferente', levando em conta que são apontadores.

Vetores e endereços

- Em C, um vetor de elementos de um tipo T é manipulado a partir do seu endereço inicial.
- Em algumas situações, um vetor de elementos do tipo T é equivalente a um endereço de uma variável desse tipo.

Vetores e endereços

- Um exemplo:

```
char a[] = "alfabeto";
```

é equivalente a

```
char *a = "alfabeto";
```

Nos dois casos, é possível acessar os elementos do vetor da mesma forma:

```
a[0] = "A";
char x = a[4];
```

Outro exemplo:

```
/* determina o máximo e o mínimo de um vetor v */
void minMax(int *v, int n, int* max, int* min){
  int i;
  *max = 0; *min = 0;
  for(i=1; i < n; i++){
    if(v[i] > v[*max]) *max = i;
    if(v[i] < v[*min]) *min = i;
  }
}
int main(){
  int i,kMax,kMin,tab[5][25];
  ...
  minMax(tab[i],25,&kMax,&kMin);
  ...
}
```

Tipos de dados definidos pelo usuário

Diretiva typedef :

```
typedef struct {
   int RA;
   char nome[25];
   char curso;
} aluno;
...
aluno a1, a2;
...
```

Define o tipo *aluno* como sendo uma estrutura. *RA*, *nome* e *curso* são *campos* dessa estrutura.

Declara as variáveis *a1* e *a2*, do tipo *aluno*.

Tipos de dados definidos pelo usuário (cont)

```
#include <strings.h>
#include <stdio.h>
int main() {
  ...
  a1.RA = 690067;
  strcpy(a1.nome,"Jose da Silva");
  a1.curso = 'c';
  printf(" RA: %d nome: %s curso: %c\n",
         a1.RA, a1.nome, a1.curso
         );
```

Mais exemplos

```
typedef int* intptr;
typedef char tnome[30];
typedef enum {
          seg,ter,qua,qui,sex,sab,dom
        } dia;
typedef aluno classe[50];
...
intptr p1;
classe Inf512;
dia hoje;
...
```

Variáveis locais a uma função

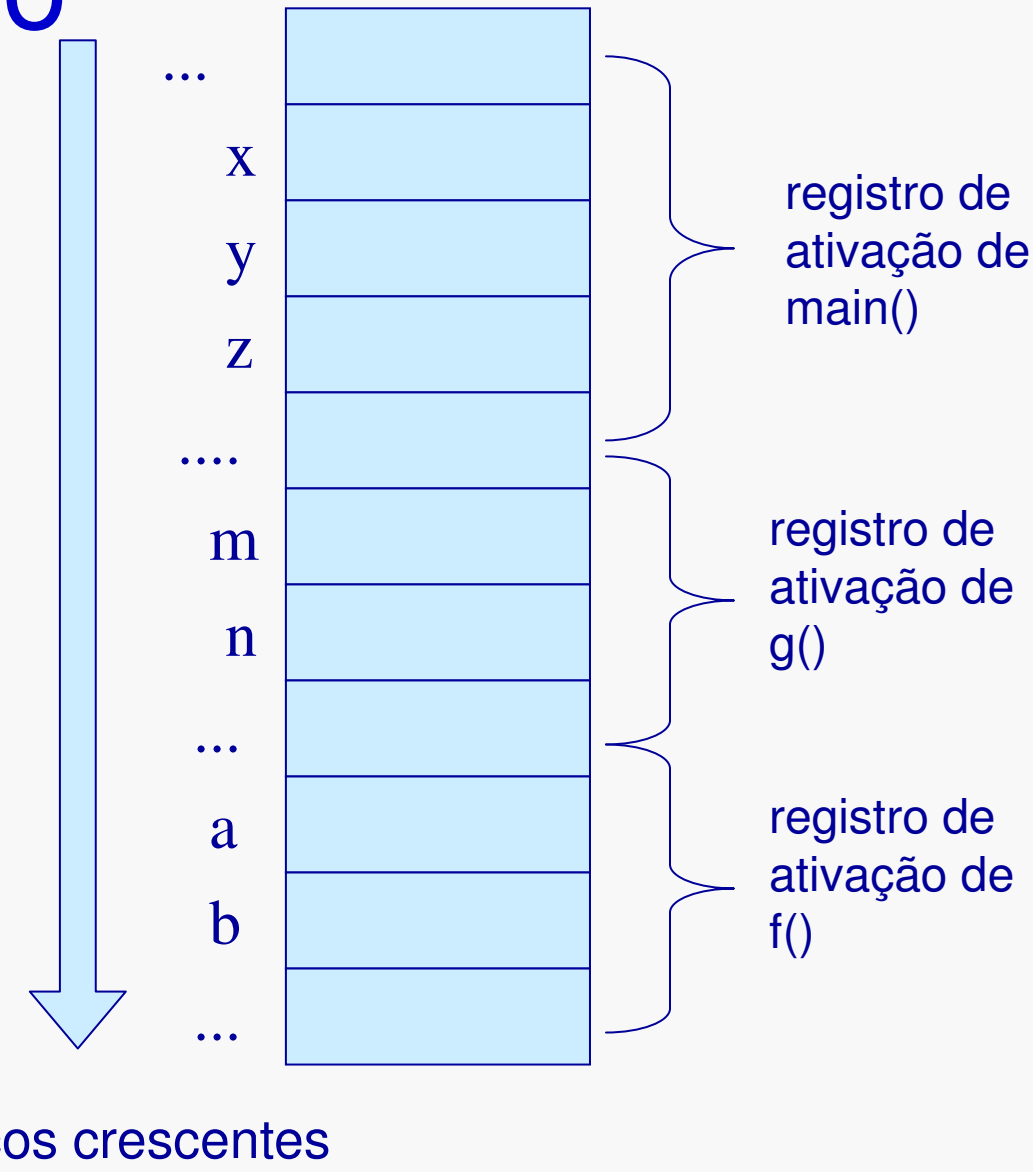
- Numa linguagem de programação típica as variáveis locais a uma função ou procedimento, são criadas no instante da chamada e destruídas antes do retorno.
- Conseqüência: os endereços das variáveis locais só são conhecidos durante a execução da mesma.

Registro de ativação

- O conjunto de posições de memória associados a uma chamada de função ou procedimento (variáveis e valores usados para controle) é chamado de ***registro de ativação.***
- O registro de ativação é criado na chamada da função e destruído ao final da sua execução.

Um exemplo

```
#include <stdio.h>
void f(){
   char a,b;
   ...
}
void g(){
   char m,n;
   ...
   f();
   ...
}
int main(){
   char x,y,z;
   ...
   g();
   ...
}
```



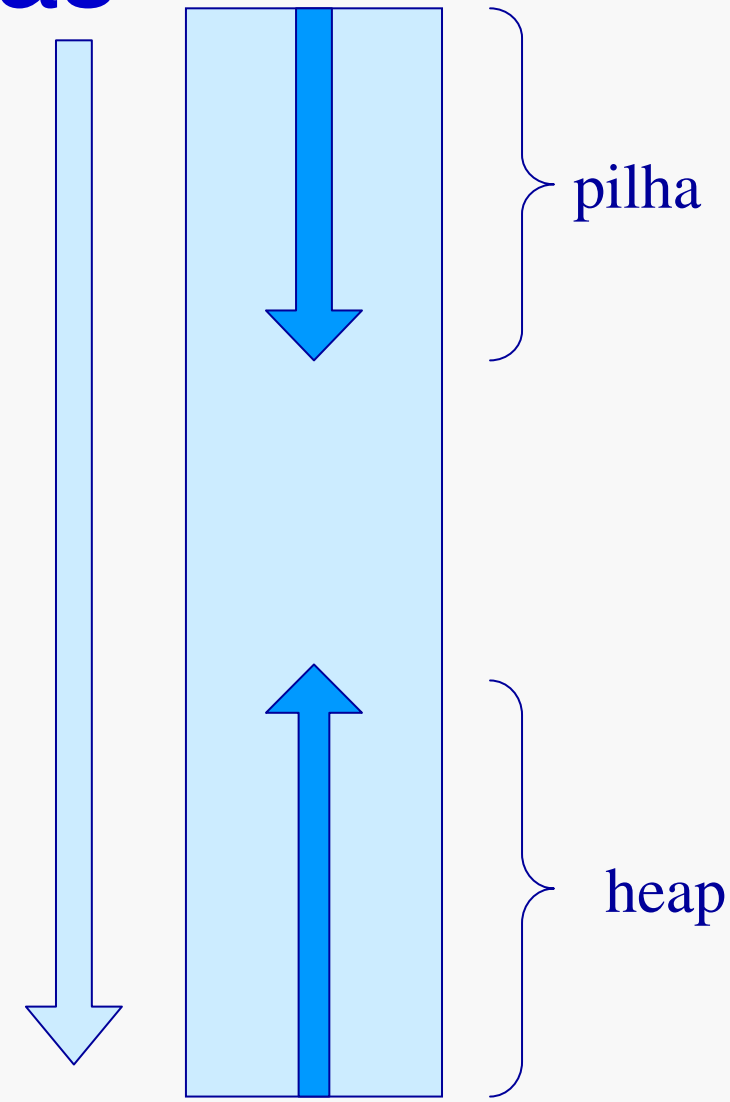
Exercício

```
#include <stdio.h>
int fat(int n){
  if(n==0) return 1;
  return n*fat(n-1);
}
int main(){
  int x;
  x = fat(3);
}
```

- Como fica o registro de ativação da função fat(), neste caso ?

Variáveis alocadas dinamicamente

```
...
char* p;
p = (char*)malloc(20);
...

var p = ^array[1..20]
        of char;
...
new(p);
...
```



Alocação dinâmica de memória

- A função de biblioteca malloc pode ser usada para alocar uma área de memória no heap.
- A sua definição (na biblioteca stdlib) é a seguinte (*size* é o tamanho, em bytes, da área a ser alocada:

```
void* malloc(int size);
```

Um exemplo

```
...
typedef struct {
  int RA;
  char nome[25];
  char curso;
} aluno;
typedef aluno* ap_aluno;
ap_aluno p;
...
p = (ap_aluno)malloc(sizeof(aluno));
p->RA = 690067;
p->curso = 'c';
strcpy(p->nome,"Joao da Silva");
...
```

Como malloc retorna um valor do tipo void*, é necessário converte-lo para ap_aluno (*type casting*).

calloc

A função calloc também pode ser usada para alocação de memória no heap.

- a área de memória alocada é preenchida com zeros.
- um parâmetro adicional indica o número de 'blocos' a ser alocado.

```
aluno classe[20] = (aluno*) calloc(20,sizeof(aluno));
```

Liberação de memória

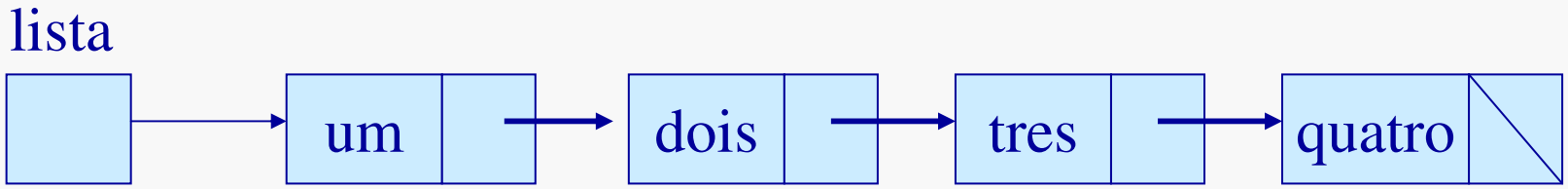
- A função free é usada para liberar uma área de memória alocada através de malloc ou calloc .
- Ela tem como parâmetro o apontador para a área a ser liberada.

```
free(classe);
free(p);
```

Listas ligadas

- cada elemento da lista contém um apontador para o próximo elemento.
- a lista é manipulada através de um apontador para o primeiro elemento.
- os elementos da lista podem ser alocados dinamicamente, de acordo com a necessidade.
- a posição física dos elementos é irrelevante.

Listas ligadas



- cada elemento tem um apontador para o próximo.
- o 'apontador da lista' aponta para o primeiro elemento.

Um exemplo (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct aluno* ap_aluno;

typedef struct aluno {
  int RA;
  char nome[30];
  char curso;
  ap_aluno prox;
} aluno;
```

apontador para a estrutura aluno, definida adiante

O nome da estrutura é necessário início da declaração porque ap_aluno se refere a ele. Ele também é necessário ao final para dar nome à estrutura.

Um exemplo (2)

```
/* cria uma estrutura do tipo aluno */
ap_aluno novoAluno(int ra, char* n, char c){
  ap_aluno p;
  p = (ap_aluno)malloc(sizeof(aluno));
  p->RA = ra;
  strcpy(p->nome,n);
  p->curso = c;
  p->prox = NULL;
  return p;
}
/* insere elemento ao final da lista */
void insere1(ap_aluno* p, int ra, char* n, char c){
  ap_aluno q = novoAluno(ra,n,c);
  q->prox = *p;
  *p = q;
}
```

Um exemplo (3)

```
/* escreve o conteudo de uma estrutura aluno */
void escreveAluno(ap_aluno ap){
  printf("RA:%d nome:%s curso:%c\n",
          ap->RA,
          ap->nome,
          ap->curso
        );
}

/* percorre a lista, escrevendo cada elemento */
void escreveLista(ap_aluno lista){
  ap_aluno p = lista;
  while (p != NULL) {
     escreveAluno(p);
     p = p->prox;
  }
```

Um exemplo (4)

```
/* libera toda a lista */
void liberaLista(ap_aluno* lista){
  ap_aluno p;
  while(*lista != NULL){
   p = *lista;
   *lista = (*lista)->prox;
   free(p);
  }
}
```

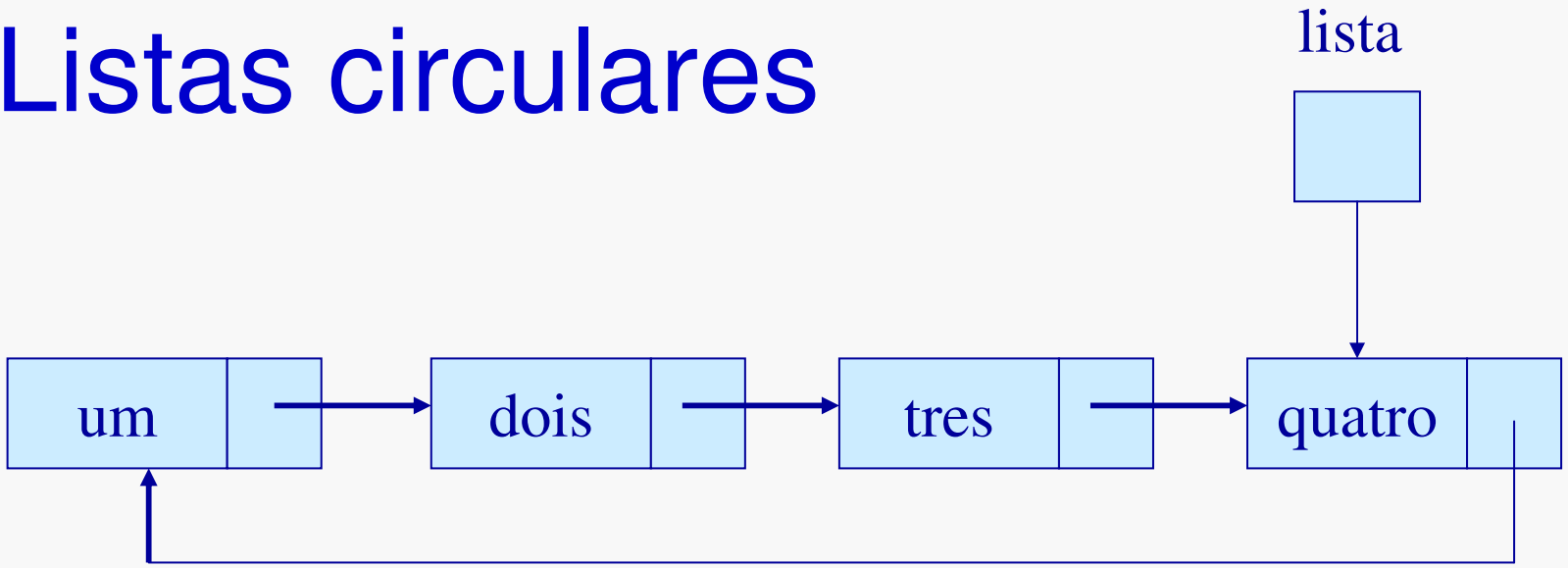
Um exemplo (4)

```
/* programa principal */
int main(void){
  /* apontador para a lista */
  ap_aluno lista = NULL;

  /* construção da lista */
  insere1(&lista,690067,"Joao da Silva",'c');
  insere1(&lista,690001,"Alberto Almeida",'c');
  insere1(&lista,690012,"Benedito Belmiro",'e');
  insere1(&lista,690015,"Carlos Collares",'m');

  escreveLista(lista);
  liberaLista(&lista);
  return 0;
}
```

Inserção ao final da lista

```
/* versão 1 - recursiva */
void insere2(ap_aluno* p, int ra, char* n, char c){
  if(*p == NULL) insere1(p,ra,n,c);
  else insere2(&((*p)->prox),ra,n,c);
}

/* versão 2 - não recursiva */
void insere2(ap_aluno* p, int ra, char* n, char c){
  ap_aluno q = *p;
  if(*p == NULL) insere1(p,ra,n,c);
  else {
   while(q->prox != NULL) q = q->prox;
   insere1(&(q->prox),ra,n,c);
  }
}
```

Listas circulares



- o último elemento aponta para o primeiro.
- o 'apontador da lista' aponta para o último elemento.

Exemplo (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef char str30[30];
typedef struct no* apno;
typedef struct no {
                str30 info;
          apno prox;
        } no;
```

Exemplo (2)

```
/* inserçao ao final da lista apontada por *p */
void insereF(apno *p, apno q){
  if(*p==NULL) q->prox = q;
  else { q ->prox = (*p)->prox;  (*p)->prox = q;
   }
  *p = q;
}

/* inserção no início da lista apontada por *p */
void insereI(apno *p, apno q){
  if(*p==NULL) { q->prox = q; *p = q; }
  else { q ->prox = (*p)->prox;  (*p)->prox = q;
   }
}
```

Exemplo (3)

```
/* remove o primeiro elemento da lista */
apno removeI(apno* p){
  apno q;
  if(*p == NULL) return NULL;
  q = (*p)->prox;
  if(q == *p) *p = NULL;
  else (*p)->prox = q->prox;
  q->prox = NULL;
  return q;
}
```

Exemplo (4)

```
/* escreve o conteúdo de um nó */
void escreveNo(apno p){
  printf("info:%s\n",p->info);
}

/* percorre a lista escrevendo seu conteúdo */
void escreveLista(apno p){
  apno q,r;
  if(p != NULL) {
   q = p->prox; r = q;
   do{
    escreveNo(q);
    q = q->prox;
   }while (r != q);
  }
}
```

Exemplo (5)

```
/* libera os nós de uma lista */
void liberaLista(apno* p){
  apno q;
  while (*p != NULL){
  q = removeI(p);
  free(q);
  }
}
```