Ponteiros

Rômulo César Silva

Unioeste

Abril de 2016

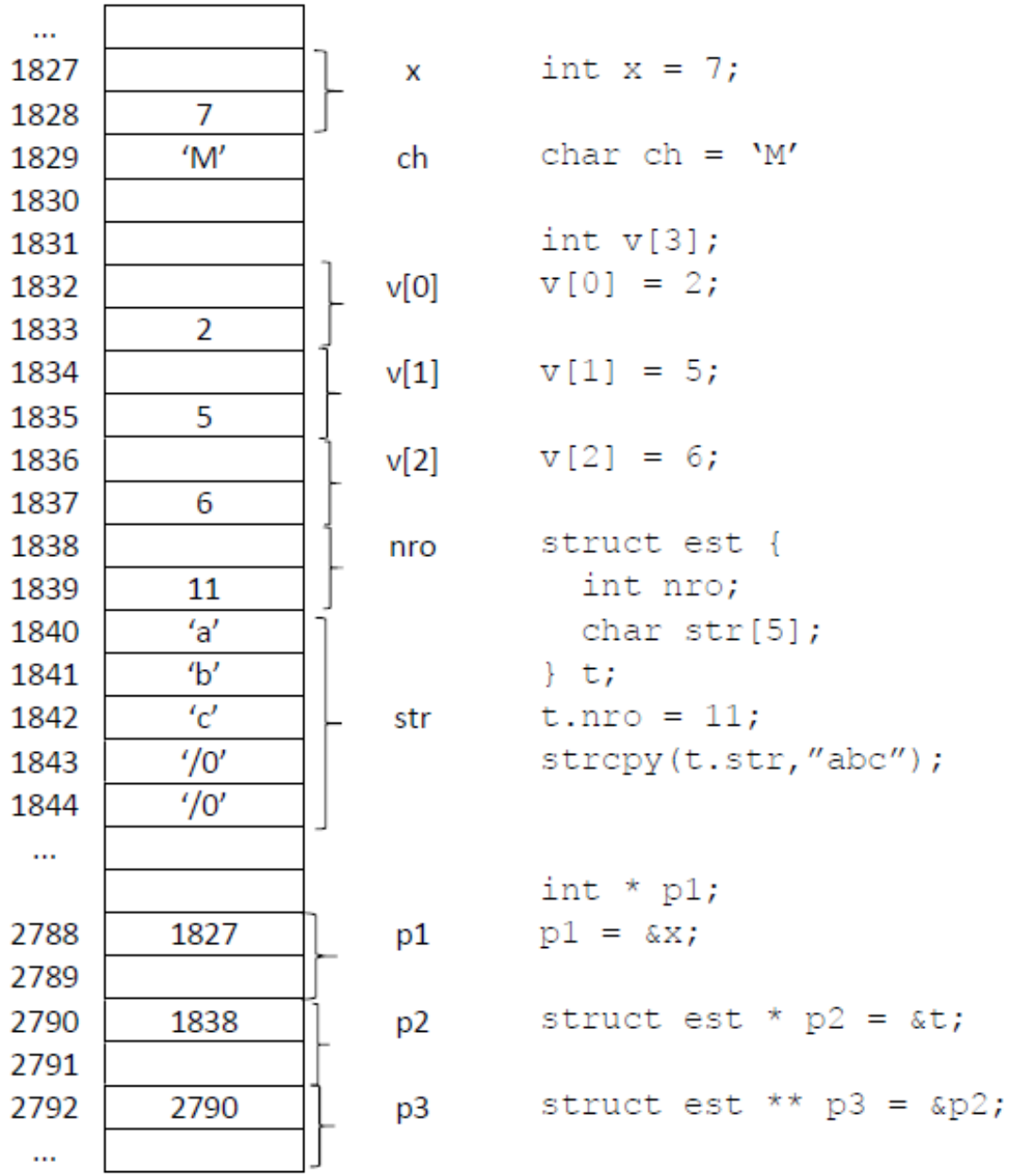
unioeste
Universidade Estadual do Oeste do Paraná

Organização da Memória de um Programa

A memória RAM de um computador é uma sequência bytes numerados sequencialmente, ao modo de uma tabela.

- cada byte armazenado pode ter um de 256 valores possíveis
- o número de um byte é o seu *endereço*
- cada objeto na memória do computador ocupa um certo número de bytes consecutivos
- a quantidade de bytes que cada tipo ocupa varia de acordo com a linguagem de programação e arquitetura/sistema operacional
- quando um objeto ocupa mais de um byte, o seu endereço é onde ele começa

Organização da Memória de um Programa



Ponteiro

Uma variável de tipo **ponteiro** armazena um endereço de memória.

- quando uma variável do tipo ponteiro contém um valor, dize-se que o ponteiro aponta para o endereço de memória.

Ponteiros na linguagem C

- há um tipo distinto para cada tipo básico C, mesmo todos os endereços tendo o mesmo tamanho
- o tipo de um ponteiro indica quantos bytes o objeto apontado ocupa na memória e como esses bytes devem ser interpretados

Exemplos de declaração:

```
int * p1; // ponteiro para int
char * ch; // ponteiro para char
typedef struct {
  int matricula;
  char nome[30];
  } aluno;
aluno * a1; // ponteiro para estrutura aluno
```

Ponteiros na linguagem C

- a simples declaração não causa alocação de memória para o objeto que será apontado
- a simples declaração também não inicializa o ponteiro. Assim, ele estará apontando para um endereço qualquer (**lixo**);
- a alocação de memória e consequentemente inicialização do ponteiro deve ser feita com o uso das funções `malloc` e `sizeof`
  - `malloc` aloca uma quantidade de bytes passada como parâmetro
  - `sizeof` calcula a quantidade de bytes necessários para um tipo
- o acesso ao conteúdo do ponteiro (objeto apontado) deve ser feito pelo operador `*` (não confundir com multiplicação!)

Ponteiros na linguagem C

Exemplos:

```
int * p1;
p1 = (int*) malloc(sizeof(int)); // aloca um ponteiro
*p1 = 7;                         // para um int
typedef struct {
  int matricula;
  char nome[30];
} aluno;
aluno * a1 = (aluno*) malloc(sizeof(aluno));
(*a1).matricula = 1234;
strcpy((*a1).nome,"João da Silva");
```

Ponteiros na linguagem C

- no caso de estruturas, pode-se fazer uso do operador -> para acessar os membros

```
typedef struct {
  int matricula;
  char nome[30];
} aluno;
aluno * a1 = (aluno*) malloc(sizeof(aluno));
a1->matricula = 1234;
strcpy(a1->nome,"João da Silva");
```

Ponteiros na linguagem C

- a liberação da área de memória alocada dinamicamente usando `malloc` é feita através da função `free`
- além disso, deve-se atribuir `NULL` após a liberação

Exemplo:

```
int * p = (int*) malloc(sizeof(int));
*p = 11;
aluno * a1 = (aluno*) malloc(sizeof(aluno));
a1->matricula = 1234;
strcpy(a1->me,"João da Silva");
...
free(p); //libera área apontada por p
p = NULL;
free(a1); //libera área apontada por a1;
a1 = NULL;
```

Ponteiros na linguagem C

ATENÇÃO!!

A maioria dos erros em programas na linguagem C ocorrem por:

- inicialização inadequada de ponteiros
- falta de liberação de memória não mais referenciada por ponteiros válidos (*memory leak*)

Ponteiros na linguagem C

- a obtenção do endereço de uma variável previamente declarada pode ser obtido pelo operador &

Exemplo:

```
int n = 7;
int * ptr  = &n; // ptr contém o endereço de n
*ptr = 11; // agora n vale 11
```

Vetores e Matrizes na linguagem C

- uma variável declarada como vetor em C representa um ponteiro para o início do armazenamento dos elementos
- dado um vetor `v`, e um inteiro `i`, as seguintes formas de acesso são válidas:
  - `v[i]` e `*(v+i)` acessam o valor do elemento na posição `i`
  - `&v[i]` e `v+i` referem-se ao endereço do elemento `i`

Exemplo:

```
int v[3]; // vetor de 3 elementos;
v[0] = 37;
*(v+1) = 7; // v[1] contém o valor 7
v[2] = 9;
printf("endereço do elemento de índice 2: %p", v+2);
```

Vetores e Matrizes na linguagem C

Aritmética de Ponteiros

Exemplo:

```
int * v = (int*) malloc(5*sizeof(int)); //vetor de 5
                                        //elementos;
int * ptr = v; // ptr aponta p/ v[0]
*ptr = 7; // v[0] = 7
ptr++;
*ptr = 11; //v[1] = 11
ptr++;
*ptr = 29; //v[2] = 29
ptr--;
*ptr = 17; //v[1] = 17
```

Vetores e Matrizes na linguagem C

Matrizes bidimensionais são implementadas como vetores de vetores.

Exemplo:

```
int ** mat; // ponteiro para ponteiro
int m = 5; // número de linhas
int n = 7; // número de colunas
int i,j;
mat = (int**) malloc(m*sizeof(int*));  // aloca vetor
                                       // de ponteiros
for (i = 0; i < m; i++)
   mat[i] = (int*) malloc(n*sizeof(int)); // aloca vetor
                                          // de inteiros
for (i = 0; i < m; i++)
  for(j = 0; j < n; j++)
     printf("%d ",mat[i][j]); //acessa mat[i,j]
```

Bibliografia I

[Feofiloff 2009] Paulo Feofiloff.
Algoritmos em linguagem C. Elsivier, Rio de Janeiro, 2009.