Dicas PET-Tele

"Conceitos básicos no uso de ponteiros em C"

Programa de Educação Tutorial

Engenharia de Telecomunicações

PETele))

Universidade Federal Fluminense

Niterói-RJ

Novembro / 2009

Conceitos básicos no uso de ponteiros em C

Mathyan Motta Beppu
Alexandre Santos de la Vega

23/11/2009

1 Introdução

Esse documento tem por objetivo elucidar alguns aspectos básicos na utilização de ponteiros na linguagem de programação C.

Inicialmente, é apresentada a simbologia utilizada no documento. Em seguida, são apresentadas as operações mais comuns envolvendo ponteiros. Por fim, é apresentado um exemplo de programa, bem como uma análise passo-a-passo do mesmo.

2 Simbologia

A Tabela 1 apresenta os símbolos que serão utilizados nas seções seguintes.

Símbolo	Significado
$L \longleftarrow V$	Local L recebe valor V
$NULL$	Endereço de memória inválido (em C: $NULL = 0$)
End	Endereço de memória End
$[End]$	Conteúdo do endereço de memória End
&	Operador de endereço
$*$	Operador de dereferenciação ou indirecionamento

Tabela 1: Símbolos utilizados nesse documento.

3 Operações comuns

A Tabela 2 apresenta as operações comumente encontradas no emprego de ponteiros.

Deve ser lembrado que, independentemente do tipo de dado armazenado, o mesmo pode ocupar várias posições consecutivas de memória. Assim, a expressão "endereço de memória associado com a variável v" significa o endereço base (inicial) das posições ocupadas.

Normalmente, não se utiliza uma atribuição direta de endereço, da forma $ptr = (tipo *)End$. O que deve ser feito é requisitar uma área de memória ao Sistema Operacional (SO), através de uma função específica (**alloc** ou **malloc**). Caso o SO disponibilize a área de memória requisitada, a função retorna o endereço base da área reservada. Porém, para não sobrecarregar o texto, será usada a expressão de atribuição direta de endereço.

Operação	Significado
$\&v$	endereço de memória associado com a variável v
$\&p$	endereço de memória associado com a variável p
v	$[\&v]$ ou conteúdo do endereço de memória associado com a variável v
p	$[\&p]$ ou conteúdo do endereço de memória associado com a variável p
$*p$	$[[\&p]]$ ou conteúdo do endereço de memória que está armazenado no conteúdo do endereço de memória associado com a variável p
$int\ v$	o conteúdo do endereço de memória associado com a variável v é um inteiro
$int\ *p$	o conteúdo do endereço de memória associado com a variável p é um endereço de memória cujo conteúdo é um inteiro
$v = 10$	$[\&v] \leftarrow 10$
$v2 = v1$	$[\&v2] \leftarrow [\&v1]$
$p = NULL$	$[\&p] \leftarrow 0$
$p = (int *)\ End$	$[\&p] \leftarrow End$
$p2 = p1$	$[\&p2] \leftarrow [\&p1]$
$p = \&v$	$[\&p] \leftarrow \&v$
$*p = 5$	$[[\&p]] \leftarrow 5$
$v = *p$	$[\&v] \leftarrow [[\&p]]$
$*p2 = *p1$	$[[\&p2]] \leftarrow [[\&p1]]$
$f(v)$	$f([\&v])$
$f(p)$	$f([\&p])$

Tabela 2: Operações comuns envolvendo ponteiros em C.

4 Exemplo de programa

Como ilustração para as operações citadas, será usado o programa abaixo:

```
/* Inclusão de arquivos contendo definições da linguagem */

01  #include <stdio.h>
02  #include <stdlib.h>
03  #include <malloc.h>
04
05  int main(void)
06  {
07  /* Criação de variáveis */
08  int *pi1, *pi2;
09  int vi1, vi2;
10
11  /* Atribuições de valores */
12   vi1 = 1;
13   vi2 = 2;
14
15  /* Imprimindo o conteúdo das variáveis vi1 e vi2 */
16   printf("vi1 = <%d>  e  vi2 = <%d>\n",vi1,vi2);
17
18   /* Atribuições de valores */
19   pi1 = &vi1;
20   pi2 = &vi2;
21
22   /* Imprimindo os endereços de memória das variáveis vi1 e vi2 */
23   printf("&vi1 = <%p>  e  &vi2 = <%p>\n",&vi1,&vi2);
24
25   /* Atribuições de valores */
26   *pi1 = 3;
27   *pi2 = 5;
28
29   /* Imprimindo o conteúdo do endereço de memória associado as variáveis
        pi1 e pi2 */
30   printf("*pi1 = <%d>  e  *pi2 = <%d>\n",*pi1,*pi2);
31




32   /* Alocação dinâmica de memória */
33   pi1 = (int *) malloc(sizeof(int));
34   pi2 = (int *) malloc(sizeof(int));
35
36   /* Imprimindo o novo conteúdo de endereço de memória */
37   printf("&pi2 = <%p>  e  &pi2 = <%p>\n",&pi1,&pi2);
38
```

```
39   /* Atribuições de valores */
40   *pi1 = 7;
41   *pi2 = 11;
42
43   /* Imprimindo o conteúdo do endereço de memória associado as variáveis
        pi1 e pi2 */
44   printf("*pi1 = <%d>  e  *pi2 = <%d>\n",*pi1,*pi2);
45
46   /* Atribuições de valores */
47   vi1 = *pi1;
48   vi2 = *pi2;
49
50   /*Imprimindo o conteúdo das variáveis  vi1 e vi2*/
51   printf("vi1 = <%d>  e  vi2 = <%d>\n",vi1,vi2);
52
53   /* Atribuições de valores */
54   pi2 = pi1;
55
56   /*Imprimindo o endereço de memória da variavel pi2*/
57   printf("&pi2 = <%p>\n",&pi2);
58
59   /* Atribuições de valores */
60   *pi2 = 13;
61
62   /* Imprimindo o conteúdo da variavel pi2 */
63   printf("&pi2 = <%p>\n",pi2);
64
65  return(0);
66  }
```

5 Passo-a-passo do programa usando Tabelas

1. Supondo-se que o programa tenha sido compilado e, em seguida, carregado na memória para ser executado, será assumido que as linhas 08 e 09 geraram o mapa de memória da Tabela 3.a. Assim sendo, tem-se que: &pi1 = A2F5, &pi2 = 1081 , &vi1 = 8D91 e &vi2 = 3782.

2. Nas linhas 12 e 13, são atribuídos valores para vi1 e vi1 como mostra a Tabela 3.b.

3. Nas linhas 19 e 20, pi1 e pi2 recebem o endereço de memória associado com as variáveis vi1 e vi2 respectivamente,como mostra a Tabela 3.c.

4. Nas linhas 26 e 27, são atribuídos valores aos conteúdos dos endereços de memória associados as variáveis pi1 e pi2,como mostra a Tabela 3.d.

5. Nas linhas 33 e 34, é feita uma alocação dinâmica de memória para as variáveis pi1 e pi2, sendo o conteúdo do endereço de memória associado a essas variáveis um inteiro. Isso foi feito usando a função malloc().

6. Nas linhas 40 e 41, são atribuídos valores aos conteúdos dos endereços de memória das váriaveis pi1 e pi2, como mostra a Tabela 3.e.

7. Nas linhas 47 e 48, as variáveis vi1 e vi1 recebem os valores dos conteúdos dos endereços de memória das variáveis pi1 e pi2 respectivamente, como mostra a Tabela 3.f.

8. Na linha 54, pi2 recebe o valor da variável pi1, como mosta a Tabela 3.g.

9. Na linha 60, o conteúdo do endereço de memória da variável pi2 recebe o valor 13, como mostra a Tabela 3.h.

Memória	
Endereço	Conteúdo
A2F5(pi1)	
8D91(vi1)	
4A2C	
3782(vi2)	
1081(pi2)	

(a)

Memória	
Endereço	Conteúdo
A2F5(pi1)	
8D91(vi1)	1
4A2C	
3782(vi2)	2
1081(pi2)	

(b)

Memória	
Endereço	Conteúdo
8D91(pi1)	
8D91(vi1)	
4A2C	
3782(vi2)	
3782(pi2)	

(c)

Memória	
Endereço	Conteúdo
8D91(pi1)	3
8D91(vi1)	
4A2C	
3782(vi2)	
3782(pi2)	5

(d)

Memória	
Endereço	Conteúdo
8D91(pi1)	7
8D91(vi1)	
4A2C	
3782(vi2)	
3782(pi2)	11

(e)

Memória	
Endereço	Conteúdo
8D91(pi1)	
8D91(vi1)	7
4A2C	
3782(vi2)	11
3782(pi2)	

(f)

Memória	
Endereço	Conteúdo
8D91(pi1)	7
8D91(vi1)	
4A2C	
3782(vi2)	
3782(pi2)	7

(g)

Memória	
Endereço	Conteúdo
8D91(pi1)	
8D91(vi1)	
4A2C	
3782(vi2)	
3782(pi2)	13

(h)

Tabela 3: “Tabelas para análise do programa exemplo.”