MC102 Algoritmos e Programação de Computadores

Aula sobre Ponteiros – 1ª Parte

1. Objetivos

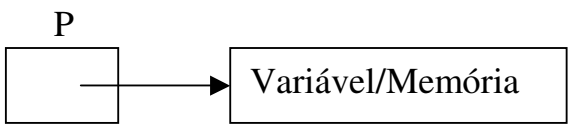
- Conceituar e ilustrar a aplicação de variáveis do tipo ponteiro, em algoritmos e programação de computadores.

2. Motivação

- O uso de ponteiros pode ser justificado em casos onde uma determinada variável seria acessada em diferentes partes do programa. Neste caso, poderiam existir vários apontadores (ponteiros) espalhados pelo programa, apontando para essa variável que deteria o conteúdo desejado (a informação). Toda e qualquer modificação no dado seria feita no conteúdo da variável que todos os apontadores fariam a referência, ou seja, diferentes partes do programa sempre estariam acessando um valor de dado atualizado.

3. Ponteiros

- Ponteiros ou apontadores são variáveis que armazenam um endereço de memória (isto é, o endereço de outras variáveis)
- Na linguagem C, cada ponteiro tem um tipo, podendo ser um ponteiro para um tipo pré-definido da linguagem ou um tipo definido pelo programador. Dessa forma, podemos ter ponteiros para inteiros, ponteiros para real, ponteiro para TipoAluno, etc
- Quando um ponteiro contém o endereço de uma variável, dizemos que o ponteiro está "apontando" para essa variável;
- Podemos representar um ponteiro P da seguinte forma:



- Neste caso, a variável ponteiro-P, lado esquerdo, está apontando para a variável-memória que está do lado direito. Internamente, a variável P contém o endereço da variável apontada

Exemplo de Declaração:

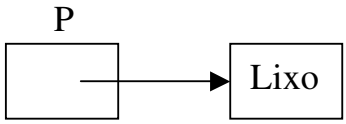
No Algoritmo:

lista_de_identificadores : *TipoPonteiro;

Na Linguagem:

lista_de_identificadores : *TipoPonteiro;

- Ao declarar um ponteiro P, o endereço inicial contido nesta variável deve ser considerado como "lixo":



Atribuindo um Endereço para um Ponteiro:

- Supondo que temos uma variável, V, e queremos que um ponteiro, P, (ponteiro para inteiro), aponte para esta variável. Para isso, usa-se o operador & para obter o endereço de V, da seguinte maneira:

  P = &V;

- Existe um endereço especial, chamado “nulo”, que serve para dizer que é um endereço nulo e não teremos nenhuma variável neste endereço de memória:

  P = nulo;

Trabalhando com a Memória Apontada pelo Ponteiro:

- Para acessar a memória que o ponteiro P está apontando, usamos o operador *, com a estrutura *P.

Exemplos de Operações no algoritmo:

Algoritmo	**Configuração da Memória**
inteiro X; inteiro *p, *q;	p (lixo) q (lixo) X (lixo)
X = 10;	p (lixo) q (lixo) X (10)
p = &X;	p (apontando para X) q (lixo) X (10)
q = p;	p (apontando para X) q (aponta também para X) X (10)
*q = *p + 10;	p (apontando para X) q (aponta também para X) X (20)
escreva (*p);	Imprime o valor 20

Exemplos de Operações na linguagem:

Programa	**Configuração da Memória**
int X; int *p, *q;	p (lixo) q (lixo) X (lixo)
X = 10;	p (lixo) q (lixo) X (10)
p = &X;	p (apontando para X) q (lixo) X (10)
q = p;	p (apontando para X) q (aponta também para X) X (10)
	p (apontando para X)

*q = *p + 10;	q (aponta também para X) X (20)
printf (“%d.\n”, *p);	Imprime o valor 20

Exemplo: aritmética de ponteiros.

```
void main(void)
{
   int vetor[] = { 10, 20, 30, 40, 50 };
   int *p1, *p2;
   int i = 100;

   p1 = &vetor[2];
   printf("%d\n", *p1);
   p2 = &i;
   printf("%d\n", *p2);
   p2 = p1;
   printf("%d\n", *p2);
}
```

Exemplo: incremento e decremento de ponteiros.

```
void main(void)
{
   int vetor[] = { 10, 20, 30, 40, 50 };
   int *p1;

   p1 = &vetor[2];
   printf("%d\n", *p1);
   p1++;
   printf("%d\n", *p1);
   p1 = p1 + 1;
   printf("%d\n", *p1);
}
```

Exemplo: uso de ponteiros e vetores (imprimindo o mesmo vetor usando formas diferentes).

```
void main(void)
{
   float v[] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0};
   int i;
   for (i=0; i<9; i++) printf("%.1f ", v[i]);
   printf("\n");
   for (i=0; i<9; i++) printf("%.1f ", *(v+i));
}
```

Exemplo: percorrendo um vetor.

```
void main(void)
{
   float v[] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0};
   int i;
   float *p;

   for (i=0; i<9; i++) printf("%.1f ", v[i]);
   printf("\n");
   for (i=0; i<9; i++) printf("%.1f ", *(v+i));
   printf("\n");
   for (i=0, p=v; i<9; i++, p++) printf("%.1f ", *p);
}
```

Exemplo: programa sobre operações com ponteiros.

```
#include <stdio.h>

void main()
{
        int x = 5;
        int y = 6;
        int* px;
        int* py;
        /* Atribuição de ponteiros */
        px = &x;
        py = &y;
        /* Comparação de ponteiros */
        if(px < py)
        printf("py-px = %u\n", py - px);
```

```
        else
        printf("px-py = %u\n", px - py);
        printf("px = %u, *px = %d, &px = %u\n", px, *px, &px);
        printf("py = %u, *py = %d, &py = %u\n", py, *py, &py);
        px++;
        printf("px = %u, *px = %d, &px = %u\n", px, *px, &px);
        py = px + 3;
        printf("py = %u, *py = %d, &py = %u\n", py, *py, &py);
        printf("py-px = %u\n", py - px);
}
```

SAÍDA

```
py-px = 1
px = 65492, *px = 5, &px = 65496
py = 65494, *py = 6, &py = 65498
px = 65494, *px = 6, &px = 65496
py = 65500, *py = -24, &py = 65498
py-px = 3
```

Exemplo: função que conta o número de caracteres de uma cadeia de caracteres.

```
int strtam(char *s);

void main(void)
{
   char *lista="1234567890";

   printf("O tamanho do string \"%s\" e %d caracteres.\n", lista, strtam(lista));
   printf("Acabou.");
}

int strtam(char *s){
   int tam=0;

   while(*(s + tam++) != '\0');
   return tam-1;
}
```

Referências

Herbert Schildt, "C Completo e Total", Makron Books.
Herbert Schildt, "C Avançado - Guia do Usuário", McGraw-Hill.
Keith Tizzard, "C for Professional Programmers", John Wiley & Sons.