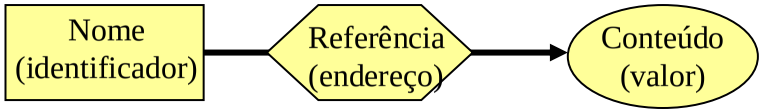
Ponteiros

Prof. Angelo Loula
UEFS

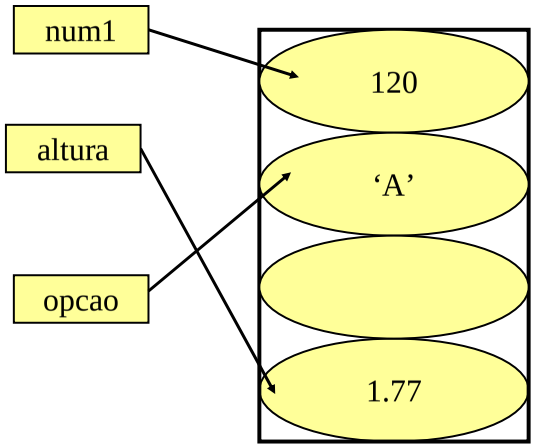
Variáveis

- Estrutura para armazenamento de dados.
  - Tipos de dados do conteúdo é especificado.
- Possui um nome para identifica-lá (*identificador*)
- Representam uma região da memória.
  - Abstração de locais de memória, referenciados por endereços.



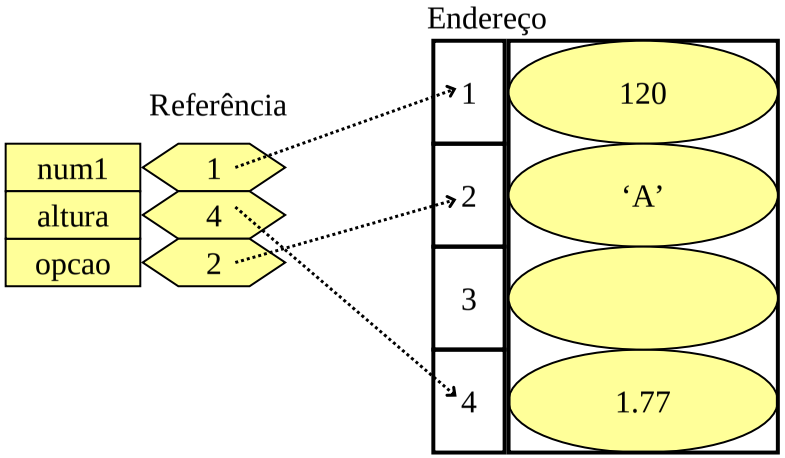
Variáveis

- Variáveis na memória



Variáveis

- Variáveis na memória



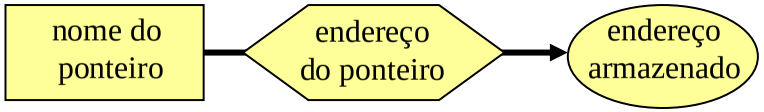
Variáveis

Limitações das variáveis

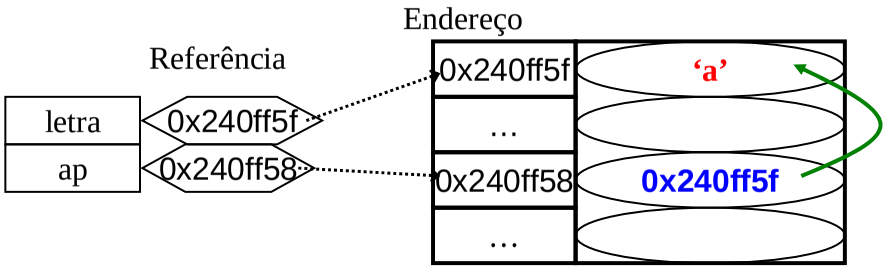
- Variáveis:
  - Restritas à declaração no código
  - Impossível criar novas variáveis durante execução
- Vetores:
  - Tamanho fixo
  - Impossível aumentar/diminuir tamanho

Ponteiros

- Ponteiro ou Apontador:
  - Tipo especial de variável
  - Número inteiro
  - Armazena um endereço de memória
- Ele **aponta** para uma posição na memória
- Endereço do ponteiro vs. Conteúdo do ponteiro



Ponteiros



Como *ap* possui o endereço da variável *letra*,
dizemos que *ap* aponta para o conteúdo da variável *letra*,
sendo possível ler e alterar o conteúdo de *letra* via o apontador *ap*.

Ponteiros

- Os ponteiros armazenam o endereço de uma informação e não a própria informação em si.
- Ponteiros acessam indiretamente um dado na memória de outra variável, em dois passos:
  - Primeiro, consulta-se a memória na posição da variável ponteiro. Neste endereço, o valor armazenado é um endereço que é usado para dar a posição de uma segunda consulta na memória, que coincide com a posição de uma variável.

Ponteiros

- Por que ponteiros?
  - Eventualmente, torna-se necessário que uma função dê acesso ao seu espaço de memória a outra função. Exemplo:

```
scanf ("%c", &letra);
```

mas e `printf ("%c", &letra);`

  - O espaço de memória utilizado por um programa pode ser reservado (e liberado) dinamicamente, via ponteiros, diferentemente do que temos feito até o momento.

Ponteiros

- Declaração:

  tipo ***** *indentificador*;

  Exemplos:

```
int *apnum;
char *apc;
```

- Utilização
  - Operador referência & (*endereço de*)
  - Operador dereferência * (*conteúdo referenciado por*)


```
#include <stdio.h>
int main () {

char letra = 'a';
/* Declaração de ponteiro para uma variável do tipo
   caracter */
char *p;

/* Atribuição de um endereço */
  p = &letra;

/* Exibe o conteúdo da variável letra */
  printf ("%c\t", *p);

/* Altera o conteúdo da variável letra */
  *p = 'b';

  printf ("%c\n", letra);
}
```



```
#include <stdio.h>
int main() {
int i = 10, *pi;
float f = 1.0, *pf;

  pi = &i;

  printf ("i = %d\n", *pi);

  pf = &f;

  printf ("f = %f\n", *pf);

  *pi = *pi + 1;
  *pf = *pf * 10;

  printf ("i = %d\n", *pi);
  printf ("f = %f\n", *pf);
}
```

Ponteiros

- Inicialização
  - Se quisermos indicar que um ponteiro não aponta para uma variável, podemos atribuir a ele um "valor nulo":

```
p = NULL;
```

  - Essa informação pode ser útil em expressões condicionais:

```
if (p == NULL) {
     …
}
```


Ponteiros

- Ponteiros e Vetores

```
int v[] = {10, 20, 30, 40, 50};
int *p;
p = v;
/* p aponta para o primeiro elemento do
  vetor */
```

  - Se quisermos o conteúdo do quarto elemento:
    1. `p[4]`
    2. `*(p+4)`

Ponteiros

- Vetor de Ponteiros

```
#include <stdio.h>
int main(){
int *v[5];
int i;
int k = 7;

  for (i = 0; i < 5; i++){
     v[i] = &i;
  }
  v[2] = &k;
  for (i = 0; i < 5; i++){
     printf ("%d ", *v[i]);
  }
}
```


Ponteiros

- Qual é a saída ?

```
int x[3] = {1, 2, 3};
int *p;

p = x;
*(p+1) = p[2];
*p = *(p+1);


printf("%d\n", p[1]);
printf("%d\n", x[1]);
printf("%d\n", p);
```