



Algoritmos e Programação de Computadores

Vetores

Prof. Lucas Boaventura
lucas.boaventura@unb.br





Introdução

- Toda informação contida está armazenada na memória do computador

Endereço

Valor

...

...

0x7ffcc203070c	0
0x7ffcc2030708	8355
0x7ffcc2030704	4
0x7ffcc2030700	-1366160

...

...





Introdução

- As variáveis são mapeadas pelo compilador para determinada(s) região(ões)

Endereço

...

Valor

...

Variável

0x7ffcc203070c	0
0x7ffcc2030708	8355
0x7ffcc2030704	4
0x7ffcc2030700	-1366160

...

...

i
num





Introdução

- `num = 5;`

Endereço

...

Valor

...

Variável

0x7ffcc203070c	0
0x7ffcc2030708	8355
0x7ffcc2030704	5
0x7ffcc2030700	-1366160

...

...

i
num





Introdução

- Desta forma, o conteúdo das variáveis são sempre únicos, cada variável pode armazenar apenas um valor por vez
- Operações subsequentes nas variáveis sobrescrevem os valores antigos: eles são perdidos “para sempre”!





Vetores

- Muitas vezes, precisamos processar diversos valores
- Armazenar em múltiplas variáveis pode tornar o código complexo, ou até mesmo ser impossível
- Ex: faça um programa que leia N valores inteiros e armazene em N variáveis





Vetores

- Para resolver esse problema, precisamos utilizar vetores de variáveis
- Utiliza-se colchetes [] junto à declaração da variável para determinar o tamanho de vetor desejado
- Declaração de um vetor de tamanho 10
 - `int vet[10];`





Vetores

- Depois da declaração, também podemos usar os colchetes `vet[n]` para acessar o n -ésimo elemento do vetor `vet`
- Atribuir o valor 5 para o membro número 2:
 - `vet[2] = 5;`





Vetores

- Atribuir o valor da variável “i” para o membro número 4 do vetor vet
 - `vet[4] = i;`
- Atribuir o valor 5 para o *i*-ésimo membro do vetor vet
 - `vet[i] = 5;`





Vetores

- Observação: os vetores em C começam com o índice 0
- Portanto, ao declarar `int vet[3]` pode-se acessar os membros:
 - `vet[0]`
 - `vet[1]`
 - `vet[2]`





Vetores

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

```
int main()
```

```
{
```

```
    int vet[10];
```

```
    vet[0] = 2;
```

```
    vet[5] = 10;
```

```
    vet[3] = -1;
```

```
    return 0;
```

```
}
```





Vetores

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

```
int main()
```

```
{
```

```
    int vet[10];
```

```
    vet[0] = 2;
```

```
    vet[5] = 10;
```

```
    vet[3] = -1;
```

```
    return 0;
```

```
}
```

Declaramos um vetor
com 10 posições

Acessamos as posições:
0 (primeira posição)
5 (sexta posição)
3 (quarta posição)





Inicialização usando for

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

```
int main()
```

```
{
```

```
    int vet[10];
```

```
    for (int i = 0; i < 10; i++)
```

```
    {
```

```
        scanf("%d", &vet[i]);
```

```
    }
```

```
    return 0;
```

```
}
```





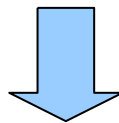
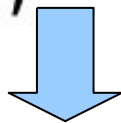
Inicialização usando for

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

```
int main()
```

```
{
```

```
    int vet[10];
```



```
    for (int i = 0; i < 10; i++)
```

```
    {
```

```
        scanf("%d", &vet[i]);
```

```
    }
```

```
    return 0;
```

```
}
```

Membros
acessados:

vet[0]

vet[1]

vet[2]

...

vet[8]

vet[9]





Inicialização usando for

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

```
int main()
```

```
{
```

```
    int vet[10];
```

```
    for (int i = 0; i < 10; i++)
```

```
    {
```

```
        vet[i] = i;
```

```
    }
```

```
    return 0;
```

```
}
```

Também é possível usar outras variáveis na atribuição. Nesse exemplo:
v[0] possui “0”
v[1] possui “1”
etc.





Lixo de Memória

- É muito importante inicializarmos as variáveis antes de utilizá-las. Em C, não existe inicialização implícita das variáveis
- O que acontece quando lemos uma variável não inicializada?

```
int main()  
{  
    int i;  
    printf("%d\n", i);  
    return 0;  
}
```





Lixo de Memória

- Lembre-se, uma variável é mapeada em uma região de memória
- Então, imprimir uma variável é ler o valor de uma determinada posição de memória
- Se a variável não foi inicializada, você não sabe qual o valor que está na posição de memória!!
- Resultado: o programa irá imprimir diferentes números “aleatórios”





Lixo de Memória

```
[user@station]$ ./exemplo_lixo_memoria  
32767
```

```
[user@station]$ ./exemplo_lixo_memoria  
32766
```

```
[user@station]$ ./exemplo_lixo_memoria  
32767
```





Lixo de Memória

- Da mesma forma, os vetores devem ser inicializados
- Mas lembre-se de que ao usar laço for, deve-se inicializar todas as posições





Alocação dinâmica

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

```
int main()  
{
```

```
    int n;
```

```
    scanf("%d", &n);
```

```
    int vet[n];
```

Declara um vetor
de tamanho n

```
    for (int i = 0; i < n; i++)
```

```
    {
```

```
        vet[i] = 0;
```

```
    }
```

```
    return 0;
```

```
}
```





Inicialização

- Também é possível inicializar um vetor os operadores {} durante a sua declaração
- Para isso, passamos uma lista de valores, em ordem, que serão atribuídos às posições do vetor

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

```
int main()
```

```
{
```

```
    int vet[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

```
    return 0;
```

```
}
```



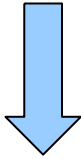


Inicialização

- Ao se inicializar dessa forma, é possível omitir o tamanho do vetor entre os colchetes
- O compilador irá calcular o tamanho necessário!

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

```
int main()  
{  
    int vet[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };  
    return 0;  
}
```





Inicialização Estática

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

```
int main()  
{
```

```
    int vet[10] = { 0, 1 };
```

```
    for (int i = 0; i < 10; i++)
```

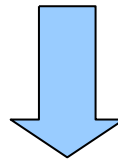
```
    {
```

```
        printf("%d\n", vet[i]);
```

```
    }
```

```
    return 0;
```

```
}
```



Posições não
declaradas são
inicializadas
como 0





Inicialização

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

```
int main()
```

```
{
```

```
    int vet[10] = { 0, 1 };
```

```
    for (int i = 0; i < 10; i++)
```

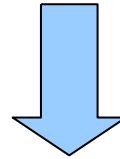
```
    {
```

```
        printf("%d\n", vet[i]);
```

```
    }
```

```
    return 0;
```

```
}
```



Posições não
declaradas são
inicializadas
como 0

Saída:

0

1

0

0

0

0

0

0

0

0





Inicialização

```
#include <stdio.h>

int main()
{
    int vet[10];

    for (int i = 0; i < 2; i++)
    {
        vet[i] = i;
    }
    for (int i = 0; i < 10; i++)
    {
        printf("%d\n", vet[i]);
    }
    return 0;
}
```

Neste caso, 2
posições
possuem 0 e 1
as outras
possuem lixo
de memória





Inicialização

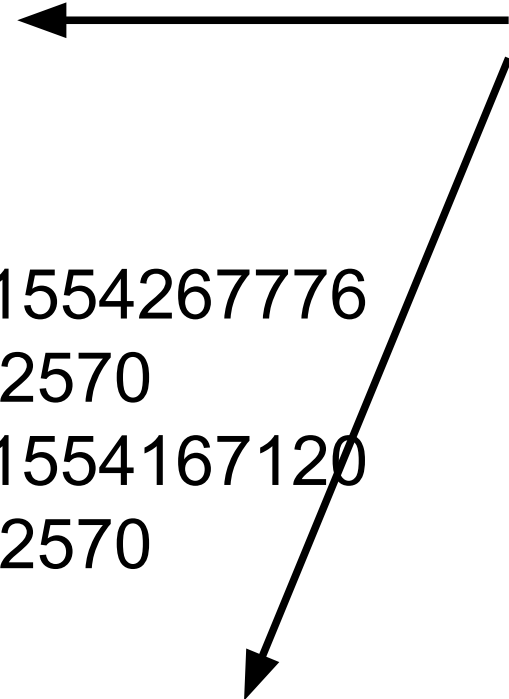
```
#include <stdio.h>

int main()
{
    int vet[10];

    for (int i = 0; i < 2; i++)
    {
        vet[i] = i;
    }
    for (int i = 0; i < 10; i++)
    {
        printf("%d\n", vet[i]);
    }
    return 0;
}
```

Números podem ser
diferentes a cada vez
que executar

0
0
0
0
-1554267776
32570
-1554167120
32570





Erro out of bounds

- Um vetor declara uma **região de memória adjacente**

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

```
int main()
```

```
{
```

```
    int vet[] = { 0, 1, 2 };
```

```
    return 0;
```

```
}
```





Erro out of bounds

- Um vetor declara uma **região de memória adjacente**

Endereço	Valor	Variável
...	...	
400	0	vet[0]
404	1	vet[1]
408	2	vet[2]
412		
...	...	





Erro out of bounds

- O que acontece quando se acessa `vet[3]`
- Neste ex., a posição de memória 412 é acessada

Endereço

Valor

Variável

...

...

400	0
404	1
408	2
412	

...

...

<code>vet[0]</code>
<code>vet[1]</code>
<code>vet[2]</code>





Erro out of bounds

- Esse comportamento é indeterminado
- O valor exato, irá depender da execução do seu programa, do sistema operacional e de outros
- Pode ser que seja lido um valor de lixo de memória
- Pode ser, que leia o valor de alguma outra variável





Erro out of bounds

- Pode ser que o seu programa não tenha permissão para ler essa posição de memória e seja finalizado pelo sistema operacional!
- Erro segmentation fault





Erro out of bounds

```
#include <stdio.h>

int main()
{
    int vet[] = { 0, 1, 2 };
    printf("%d\n", vet[250000]);
    return 0;
}
```

```
[user@station]$ ./exemplo_segmentation_fault
Segmentation fault (core dumped)
```





Dúvidas?

- lucas.boaventura@unb.br

