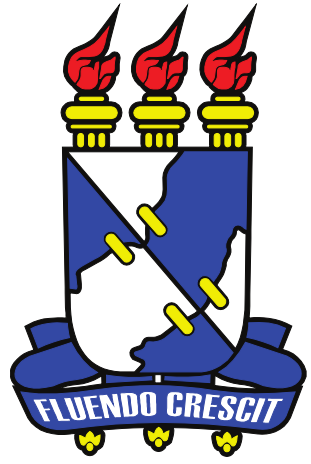


Estrutura de Dados I

Ponteiros e Alocação Dinâmica

Bruno Prado

Departamento de Computação / UFS

Introdução

- Estrutura de memória
  - Memória de 4 GB
  - Endereçamento de 32 bits
  - Palavras de 32 bits (4 bytes)

Endereço	Conteúdo			
0xFFFFFFFF	B0	B1	B2	B3
0xFFFFFFFC	B0	B1	B2	B3
⋮	⋮			
0x00000004	B0	B1	B2	B3
0x00000000	B0	B1	B2	B3

Introdução

- Little Endian
  - Primeiro byte é o menos significativo

Número 0xAABBCCDD

0	1	2	3
0xDD	0xCC	0xBB	0xAA

Introdução

- Little Endian
  - Primeiro byte é o menos significativo

Número 0xAABBCCDD

0	1	2	3
0xDD	0xCC	0xBB	0xAA

- Big Endian
  - Primeiro byte é o mais significativo

Número 0xAABBCCDD

0	1	2	3
0xAA	0xBB	0xCC	0xDD

Introdução

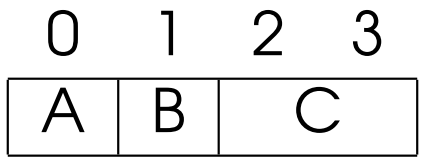
- Alinhamento de Memória
  - Não alinhado
  - Menor uso de memória
  - Acesso mais complexo

```
char A;
char B;
short int C;
```

Introdução

- Alinhamento de Memória
  - Não alinhado
  - Menor uso de memória
  - Acesso mais complexo

```
char A;
char B;
short int C;
```



Introdução

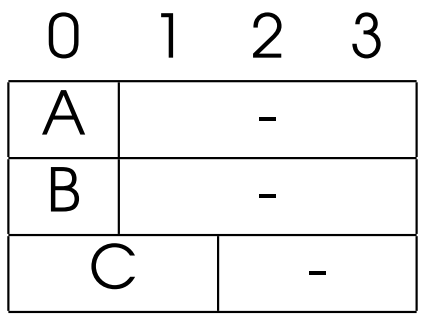
- Alinhamento de Memória
  - Alinhado
  - Maior uso de memória
  - Acesso mais simples

```
char A;
char B;
short int C;
```

Introdução

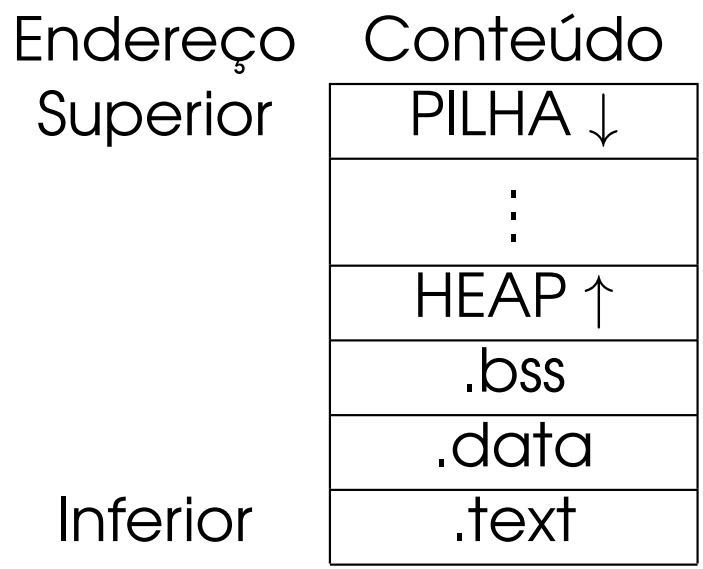
- Alinhamento de Memória
  - Alinhado
  - Maior uso de memória
  - Acesso mais simples

```
char A;
char B;
short int C;
```



Introdução

- Estruturas dinâmicas
  - Heap
    - Alocação dinâmica de memória
  - Pilha
    - Chamada de funções e passagem de parâmetros



Ponteiros

- Apontadores ou ponteiros para uma determinada região de memória
- São variáveis utilizadas para armazenamento do endereço de memória de outras variáveis
  - Declaração do tipo com *
  - Operador de endereço &

```
// Variável x tem o valor 7
int x = 7;
// Ponteiro px inicializado como nulo
int* px = NULL;
// Ponteiro recebe endereço da variável x
px = &x;
```

Ponteiros

- Formato da memória

Endereço	Variável	Conteúdo
⋮	⋮	⋮
0x80000004	px	0x80000000
0x80000000	x	0x00000007
⋮	⋮	⋮

Ponteiros

- Operando com o ponteiro
  - Exibindo endereço
  - Acessando conteúdo

```
// Variável x tem o valor 7
int x = 7;
// Ponteiro px inicializado como nulo
int* px = NULL;
// Ponteiro recebe endereço da variável x
px = &x;
// Imprimindo informações
printf("Valor de x é %d em %p\n", *px, px);
// Atualizando valor de x
*px = 3;
```

Ponteiros

- Operando com o ponteiro
  - Exibindo endereço
  - Acessando conteúdo

```
// Variável x tem o valor 7
int x = 7;
// Ponteiro px inicializado como nulo
int* px = NULL;
// Ponteiro recebe endereço da variável x
px = &x;
// Imprimindo informações
printf("Valor de x é %d em %p\n", *px, px);
// Atualizando valor de x
*px = 3;
```

> Valor de x é 7 em 0x80000000

Ponteiros

- Formato da memória

Endereço	Variável	Conteúdo
⋮	⋮	⋮
0x80000004	px	0x80000000
0x80000000	x	0x00000003
⋮	⋮	⋮

Ponteiros

- Passagem de parâmetro por valor

```
// Passando o parâmetro x por valor
f(x);
// Exibindo valor
printf("Valor é %d\n", x);
...
// Implementação da função f
void f(int valor) {
        // Exibindo o valor passado
        printf("Valor é %d\n", valor);
        // Modificando o valor
        valor = 1;
}
```

Ponteiros

- Passagem de parâmetro por valor

```
// Passando o parâmetro x por valor
f(x);
// Exibindo valor
printf("Valor é %d\n", x);
...
// Implementação da função f
void f(int valor) {
        // Exibindo o valor passado
        printf("Valor é %d\n", valor);
        // Modificando o valor
        valor = 1;
}
```

```
> Valor é 3
> Valor é 3
```

Ponteiros

- Formato da memória

Endereço	Variável	Conteúdo
⋮	⋮	⋮
0x80000004	px	0x80000000
0x80000000	x	0x00000003
⋮	⋮	⋮

Ponteiros

- Passagem de parâmetro por referência

```
// Passando o parâmetro x por referência
f(&x);
f(px);
// Exibindo valor
printf("Valor é %d\n", x);
...
// Implementação da função f
void f(int* referencia) {
        // Exibindo o valor passado
        printf("Valor é %d\n", *referencia);
        // Modificando o valor
        *referencia = 1;
}
```

Ponteiros

- Passagem de parâmetro por referência

```
// Passando o parâmetro x por referência
f(&x);
f(px);
// Exibindo valor
printf("Valor é %d\n", x);
...
// Implementação da função f
void f(int* referencia) {
        // Exibindo o valor passado
        printf("Valor é %d\n", *referencia);
        // Modificando o valor
        *referencia = 1;
}
```

```
> Valor é 3
> Valor é 1
```

Ponteiros

- Formato da memória

Endereço	Variável	Conteúdo
⋮	⋮	⋮
0x80000004	px	0x80000000
0x80000000	x	0x00000001
⋮	⋮	⋮

Ponteiros

- Modificador **const**
  - Proteger passagem por referência
  - Permissão de somente leitura

```
// Implementação da função f
void f(const int* referencia) {
        // Exibindo o valor passado
        printf("Valor é %d\n", *referencia);
        // Modificando o valor
        *referencia = 1;
}
```

Ponteiros

- Modificador **const**
  - Proteger passagem por referência
  - Permissão de somente leitura

```
// Implementação da função f
void f(const int* referencia) {
        // Exibindo o valor passado
        printf("Valor é %d\n", *referencia);
        // Modificando o valor
        *referencia = 1;
}
```

> Erro de compilação: Referência somente leitura

Ponteiros

- Ponteiro de ponteiro
  - Ponteiro que aponta para outro ponteiro

```
// Implementação de ponteiro de ponteiro
char* p = NULL;
char** pp = &p;
// Variantes
int main(int argc, char** argv);
int main(int argc, char* argv());
```

- Passagem de parâmetro por referência

Ponteiros

- Ponteiro de função
  - Ponteiro que aponta para uma função

```
// Protótipo da função fatorial
unsigned int fatorial(unsigned int valor) ;
// Implementação de ponteiro de função
unsigned int (*pf_fatorial)(unsigned int valor) = NULL;
pf_fatorial = &fatorial;
int f = (*pf_fatorial)(4);
```

- Uso em definição de API

Alocação Dinâmica

- Memória Estática
  - Variáveis de tamanho fixo previamente conhecido
  - Alocada estaticamente nos segmentos .data e .bss

Alocação Dinâmica

- Memória Estática
  - Variáveis de tamanho fixo previamente conhecido
  - Alocada estaticamente nos segmentos .data e .bss
- Memória Dinâmica
  - Variáveis de tamanho conhecido somente em tempo de execução
  - Alocada dinamicamente no heap

Alocação Dinâmica

- Medindo o tamanho em bytes das variáveis
  - Operador **sizeof**()

```
// Declarando variáveis
char c = 'B';
short int i = 12000;
float r = 12.12;
...
// Exibindo tamanho em bytes
printf("Tamanho de c é %d\n", sizeof(c));
printf("Tamanho de i é %d\n", sizeof(i));
printf("Tamanho de r é %d\n", sizeof(r));
}
```

Alocação Dinâmica

- Medindo o tamanho em bytes das variáveis
  - Operador **sizeof**()

```
// Declarando variáveis
char c = 'B';
short int i = 12000;
float r = 12.12;
...
// Exibindo tamanho em bytes
printf("Tamanho de c é %d\n", sizeof(c));
printf("Tamanho de i é %d\n", sizeof(i));
printf("Tamanho de r é %d\n", sizeof(r));
}
```

```
> Tamanho de c é 1
> Tamanho de i é 2
> Tamanho de r é 4
```

Alocação Dinâmica

- Alocando bytes de memória dinamicamente
  - Função **void*** **malloc**(size_t size)
    - Parâmetro size é o tamanho em bytes
    - Em caso de sucesso, é retornado o endereço base
    - Em caso de falha, é retornado zero (NULL)

```
// Declarando ponteiro
int* idades = NULL;
// Alocando dinamicamente
idades = (int*) malloc(50 * sizeof(int))
// Checando alocação
if(idades == NULL) printf("Erro na alocação!\n");
else printf("Vetor de idades alocado!\n");
```

Alocação Dinâmica

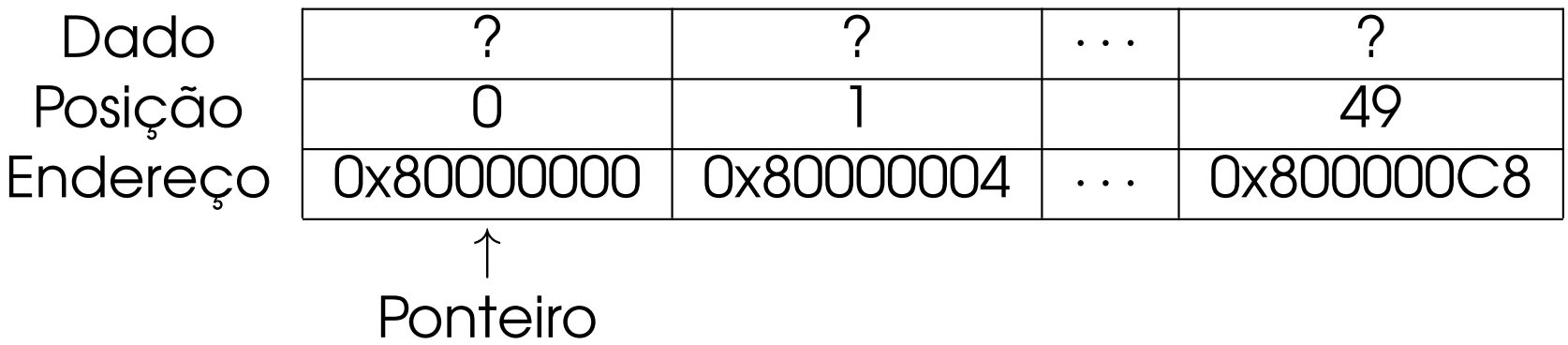
- Alocando bytes de memória dinamicamente
  - Função **void*** **malloc**(size_t size)
    - Parâmetro size é o tamanho em bytes
    - Em caso de sucesso, é retornado o endereço base
    - Em caso de falha, é retornado zero (NULL)

```
// Declarando ponteiro
int* idades = NULL;
// Alocando dinamicamente
idades = (int*) malloc(50 * sizeof(int))
// Checando alocação
if(idades == NULL) printf("Erro na alocação!\n");
else printf("Vetor de idades alocado!\n");
```

> Vetor de idades alocado!

Alocação Dinâmica

- Alocando bytes de memória dinamicamente
  - Função **void*** **malloc**(size_t size)
    - Endereço base de 0x8000000
    - Tamanho alocado de 200 bytes



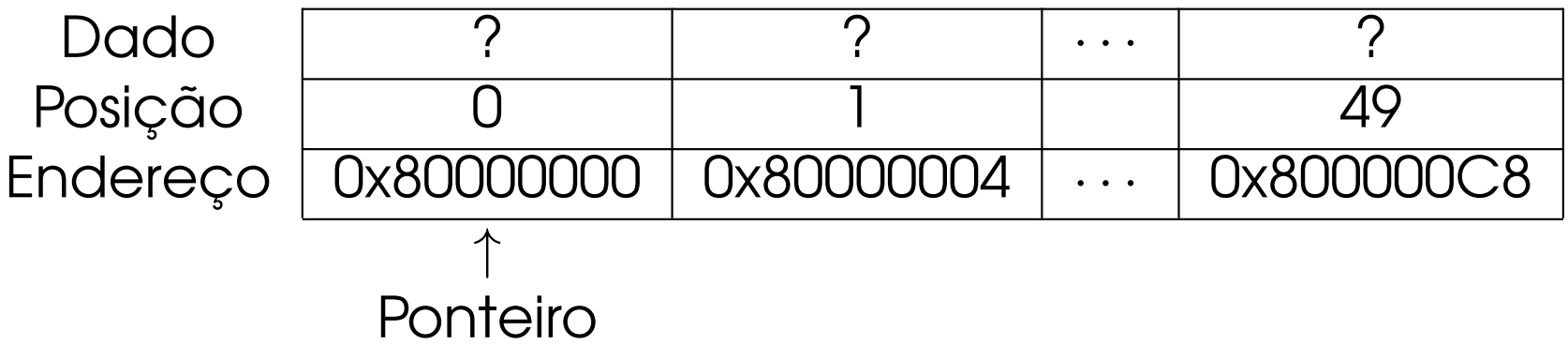
Alocação Dinâmica

- Desalocando memória
  - Função **void free**(**void*** ptr)
    - Parâmetro ptr aponta para variável dinâmica
    - O valor do ponteiro não é modificado

```
// Declarando ponteiro
int* idades = NULL;
// Alocando dinamicamente
idades = (int*) malloc(50 * sizeof(int))
...
// Desalocando memória utilizada
free(idades);
// Invalidando ponteiro
idades = NULL;
```

Alocação Dinâmica

- Desalocando memória
  - Função **void free**(**void*** ptr)
    - Endereço base de 0x8000000
    - Tamanho desalocado de 200 bytes



Alocação Dinâmica

- Alocando e inicializando memória dinamicamente
  - Função **void*** **calloc**(size_t num, size_t size)
    - Tamanho alocado = Número de Elementos (num) x Tamanho de cada elemento (size)
    - Inicializa a memória alocada com zeros

```
// Declarando ponteiro
int* idades = NULL;
// Alocando dinamicamente
idades = (int*) calloc(5, sizeof(int))
// Imprimindo idades
printf("Idades: ");
for(int i = 0; i < 5; i++) printf("%d ", idades(i));
printf("\n");
```

Alocação Dinâmica

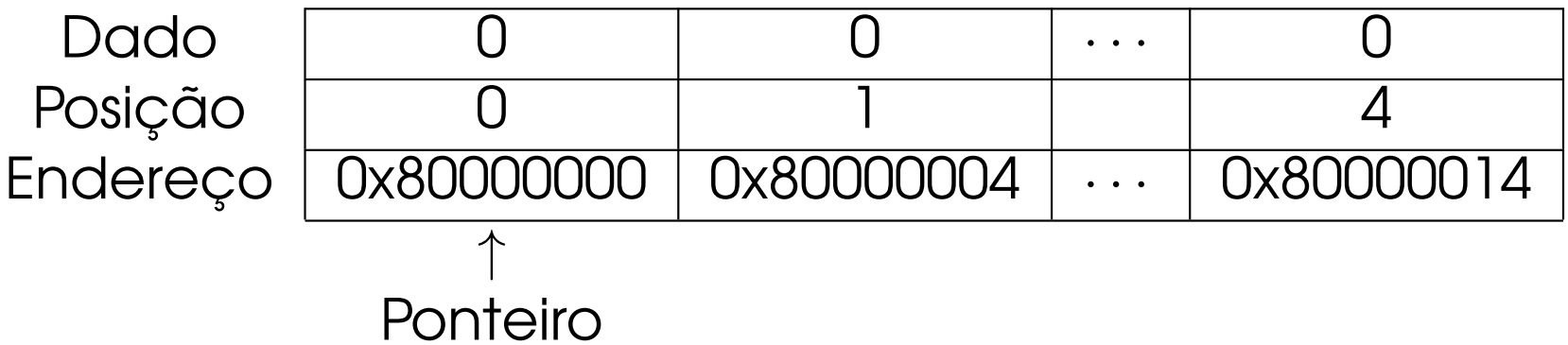
- Alocando e inicializando memória dinamicamente
  - Função **void*** **calloc**(size_t num, size_t size)
    - Tamanho alocado = Número de Elementos (num) x Tamanho de cada elemento (size)
    - Inicializa a memória alocada com zeros

```
// Declarando ponteiro
int* idades = NULL;
// Alocando dinamicamente
idades = (int*) calloc(5, sizeof(int))
// Imprimindo idades
printf("Idades: ");
for(int i = 0; i < 5; i++) printf("%d ", idades(i));
printf("\n");
```

> Idades: 0 0 0 0 0

Alocação Dinâmica

- Alocando e inicializando memória dinamicamente
  - Função **void*** **calloc**(size_t num, size_t size)
    - Endereço base de 0x8000000
    - Tamanho alocado de 20 bytes



Alocação Dinâmica

- Realoca memória dinamicamente
  - Função **void*** **realloc**(**void*** ptr, size_t size)
    - Ponteiro ptr é o ponteiro para região de memória
    - Variável size é o novo tamanho contínuo
    - Em caso de sucesso, é retornado o mesmo endereço base quando existe memória livre
    - Em caso de sucesso, é retornado um novo endereço base (quando uma nova área foi alocada)
    - Em caso de falha, é retornado zero (NULL)

```
// Declarando ponteiro
int* idades = NULL;
// Alocando dinamicamente
idades = (int*) malloc(5 * sizeof(int))
// Mais idades necessárias
int* mais_idades = NULL;
mais_idades = (int*) realloc(idades, 50 * sizeof(int))
```

Alocação Dinâmica

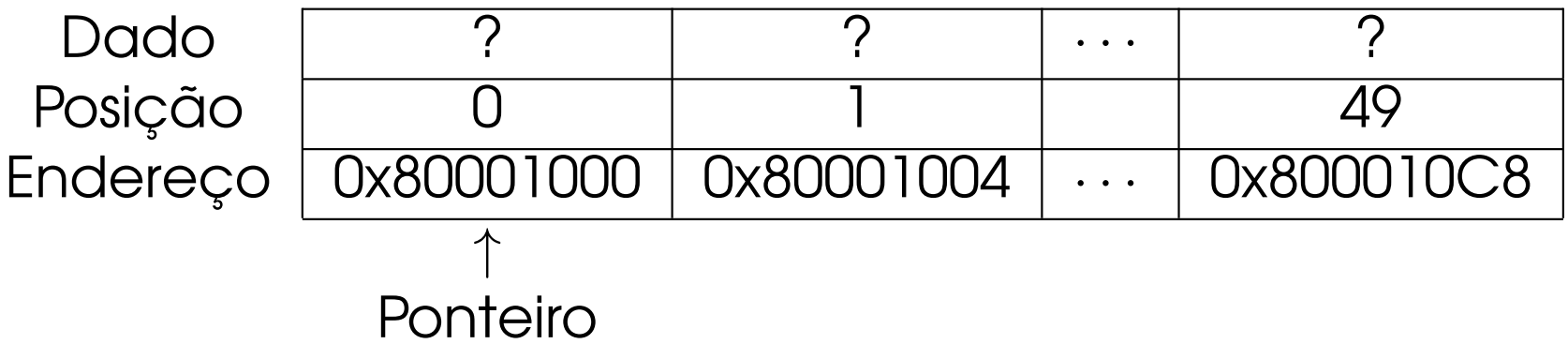
- Realoca memória dinamicamente
  - Função **void*** **realloc**(**void*** ptr, size_t size)
    - Endereço base de 0x8000000
    - Tamanho alocado de 20 bytes

Dado	?	?	...	?
Posição	0	1		4
Endereço	0x80000000	0x80000004	...	0x80000014

↑
Ponteiro

Alocação Dinâmica

- Realoca memória dinamicamente
  - Função **void*** **realloc**(**void*** ptr, size_t size)
    - Endereço base 0x80001000
    - Tamanho realocado de 200 bytes



Alocação Dinâmica

- Erros comuns
  - Acesso de endereço não alocado
    - Buffer Overflow
  - Ponteiros não inicializados ou não invalidados
    - Segmentation Fault
  - Região de memória sem nenhum ponteiro
    - Memory Leak