Introdução a Técnicas de Programação

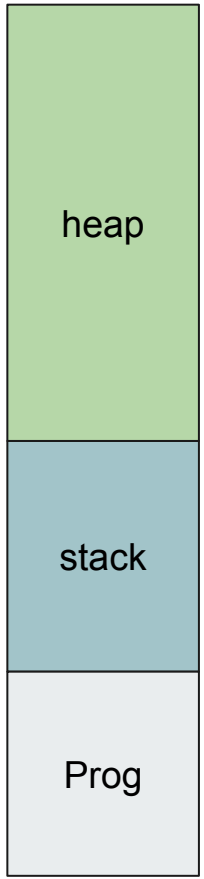
Alocação dinâmica

Prof. André Campos
DIMAp/UFRN

Memória

Quando um programa é iniciado, além do espaço para o programa na memória, o S.O. reserva dois outros espaços:

- Stack
  - Onde as variáveis são armazenadas e as rotinas empilhadas
  - Acesso rápido e eficiente (gerenciado pela CPU)
  - Não há fragmentação de espaço
  - Limitada pelo S.O.
- Heap
  - Pode ser acessado a partir de qualquer ponto do programa
  - Acesso mais lento (gerenciado pelo S.O.)
  - Pode haver fragmentação de espaço
  - Limitada pela memória do computador



Alocação estática e dinâmica

Variáveis locais são alocadas na stack.

Dizemos que é uma **alocação estática** (fixa e pré-definida na compilação)

> **Obs:** *A memória alocada para as variáveis locais, guardadas na stack, são liberadas quando a função onde são definidas termina.*

Às vezes, é necessário:

- Alocar um tamanho desconhecido durante a compilação
- Alocar um espaço muito...muito grande
- Alocar um espaço que pode ser alterado durante a execução

Precisamos de uma **alocação dinâmica**

Exemplo de necessidade de alocação dinâmica

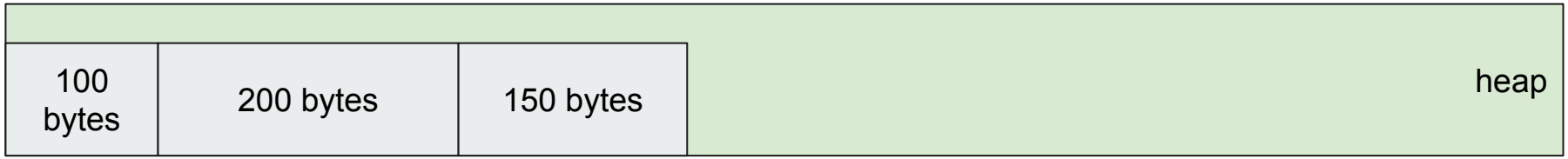
Caso exemplo

- Ler uma imagem (ex: PPM)
  - Tamanho desconhecido durante a compilação
  - Tamanho pode ser muito...muito grande
- Ampliar ou reduzir a imagem lida
  - Tamanho pode ser alterado durante a execução

Alocação dinâmica

Alocação na heap

- Definimos o tamanho do bloco de memória a ser alocado
- Liberamos o bloco quando não precisamos mais
- Precisamos guardar a referência (ponteiro) para acessar o conteúdo e liberar o bloco
- Se todo o espaço da heap for preenchido, o S.O. reserva mais espaço
  (se não houver mais memória RAM disponível, usa memória virtual)



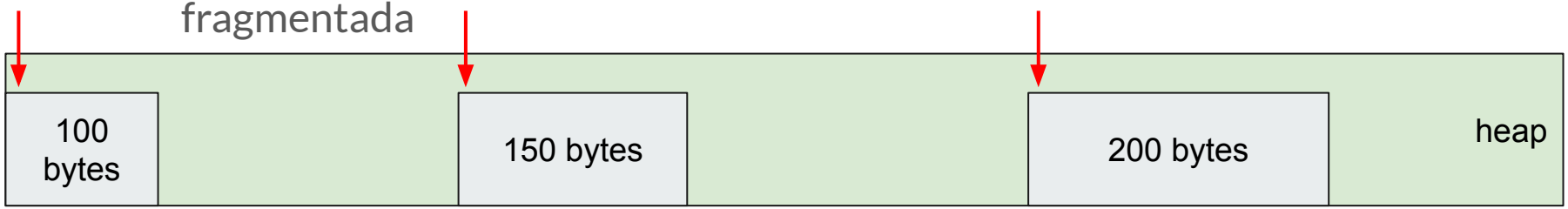
Alocação dinâmica

Acesso ao conteúdo alocado

- Precisamos guardar a referência (ponteiro) para acessar o conteúdo de um bloco alocado

Desalocação na heap

- A referência também é necessária para indicar o bloco a ser liberado
- Após várias alocações e desalocações, a memória pode ficar fragmentada



Alocação dinâmica em C++: `new` e `delete`

Operador `new` aloca memória da heap.

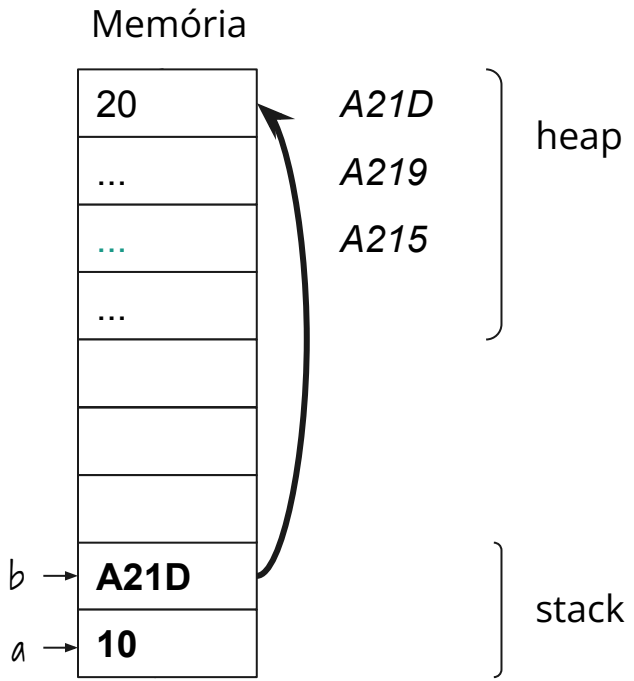
Operador `delete` **libera memória** alocada anteriormente, mas não destrói a variável do ponteiro (pode ser usada novamente).

```
int main() {
  int a;
  int *b = new int;

  a = 10;
  *b = 20;

  delete b;

  return 0;
}
```

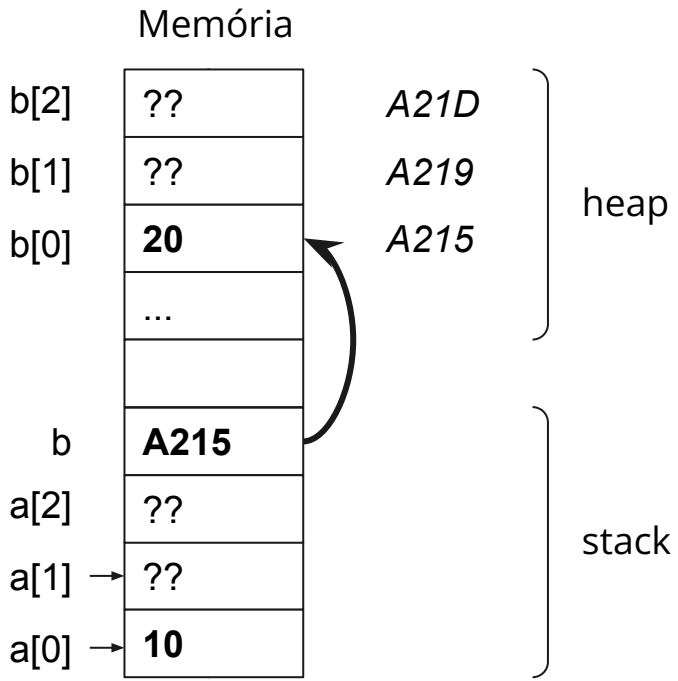


Alocação de arrays dinâmicos

Uso de [ ] para definir o tamanho do array.
Operador delete[] é usado para liberar o array.

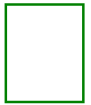
```
int main() {
  int a[3];
  int *b = new int[3];
  a[0] = 10;
  b[0] = 20;
  delete[] b;

  return 0;
}
```



Retorno de arrays

Agora, sim, podemos retornar um array (alocado dinamicamente)
O array é criado na Heap e não é removido no término da função.



```
int* criaArray(int n) {
  int array[n];
  // ...
  return array;
}
```

```
int* criaArray(int n) {
  int *array = new int[n];
  // ...
  return array;
}
```

Alocação de matrizes

Matrizes são “arrays de arrays”, portanto:

- A variável deve ser “ponteiro de ponteiro”
- A alocação e liberação da memória requer um laço.

Alocação

```
int** criaMatriz(int lin, int col) {
  int **matriz = new int*[lin];
  for (int i = 0; i < lin; ++i) {
    matriz[i] = new int[col];
  }
  return matriz;
}
```

Liberação

```
void liberaMatriz(int **matriz, int lin) {
  for (int i = 0; i < lin; ++i) {
    delete[] matriz[i];
  }
  delete[] matriz;
}
```

Outra opção para alocação de matrizes

Alocar um array de tamanho núm. de linhas x núm. de colunas

Acessar os dados do array usando a “fórmula”: `lin x num_col + col`

1	2	3	4
5	6	7	8
9	10	11	12

```
int main() {
  int num_linhas = 3;
  int num_colunas = 4;
  int *matriz = new int[num_linhas *
num_colunas];

  for (int i = 0; i < num_linhas; ++i) {
    for (int j = 0; j < num_colunas; ++j) {
      matriz[i * num_colunas + j] = i + j;
    }
  }
  return 0;
}
```

Auto-avaliação