



Quem se prepara, não para.

Estrutura de Dados

Professora: Michelle Hanne Soares de Andrade
michelle.andrade@newtonpaiva.br

Sumário:

- ✓ Alocação de Memória
- ✓ Memória Estática e Memória Dinâmica
- ✓ Ponteiros

Alocação de memória é algo que já é feito quando você declara uma variável, elemento fundamental para o funcionamento de seu programa, conforme explica Puga:

- Os dados são armazenados temporariamente em variáveis para que sejam processados de acordo com as especificações do algoritmo. [...]. Definir o tipo de dado adequado para ser armazenado em uma variável é uma questão de grande importância para garantir a resolução do problema. (PUGA, 2013, p. 34)

Ainda para Puga (2013, p. 38), “Quando um algoritmo é transcrito para uma determinada linguagem de programação as variáveis também têm a função de armazenar dados temporariamente, mas na memória RAM do computador”.

A alocação estática de memória é aquela **feita durante a declaração de uma variável**, onde ela é declarada com um tamanho específico e **não pode ser alterada em tempo de execução**.

Pode-se utilizar como exemplo de alocação estática: vetores e matrizes (estruturas estáticas).

Quando uma variável é declarada, um espaço de memória é alocado para a mesa e, quando ela é inicializada, nesse espaço é armazenado seu valor:

```
Int x;  
X=0;
```

Se o tipo da variável não for de um dos tipos primitivos, então, ela é uma referência para um objeto.

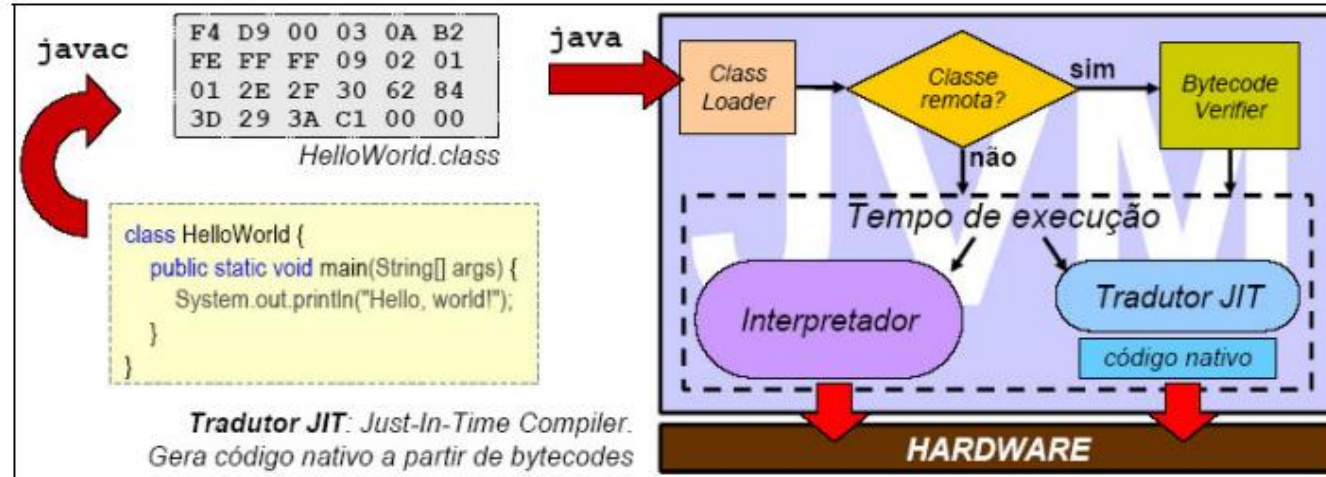
Em Java todo o processo de alocação do objeto na memória é gerenciado pela máquina virtual Java (JVM).

A máquina virtual Java (JVM) é uma máquina imaginária implementada como uma aplicação de software [JVMS]². Ela executa um código de máquina portátil (chamado de Java bytecode) armazenado em um formato de arquivo chamado de class file format (formato de arquivo class). Um arquivo em formato class geralmente³ é gerado como resultado de uma compilação de código-fonte Java.

Em Java todo o processo de alocação do objeto na memória é gerenciado pela máquina virtual Java (JVM).

A máquina virtual Java (JVM) é uma máquina imaginária implementada como uma aplicação de software [JVMS]2. Ela executa um código de máquina portátil (chamado de Java bytecode) armazenado em um formato de arquivo chamado de class file format (formato de arquivo class). Um arquivo em formato class geralmente3 é gerado como resultado de uma compilação de código-fonte Java.

Máquina Virtual Java

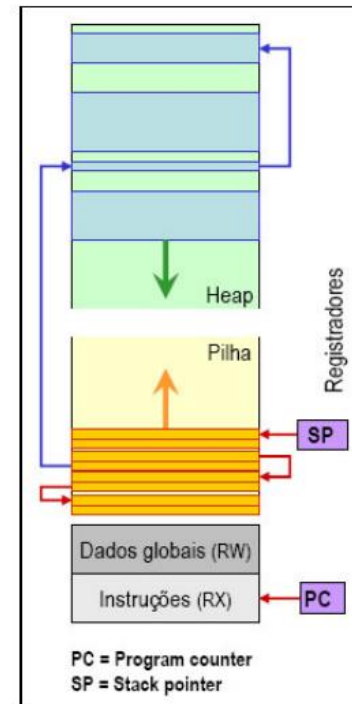


A pilha, o heap e a JVM Existem linguagens em que a alocação de **memória é trivial**, e não requer gerenciamento complexo. As principais estratégias são:

- ◆ Alocação estática: áreas de memória são alocadas antes do início do programa; não permite mudanças nas estruturas de dados em tempo de execução (ex: Fortran)
- ◆ Alocação linear: memória alocada em fila ou em pilha; não permite remoção de objetos fora da ordem de criação (ex: Forth)
- ◆ Alocação dinâmica: permite liberdade de criação e remoção em ordem arbitrária; requer gerência complexa do espaço ocupado e identificação dos espaços livres (ex: Java, C++) .

Alocação Estática de Memória

Java utiliza alocação dinâmica (heap) para objetos e alocação linear (pilha) para procedimentos sequenciais, mas todo o gerenciamento é feito automaticamente.





Ponteiros

Ponteiros guardam endereços de memória.

O C funciona assim: você anota o endereço de algo numa variável ponteiro para depois usar.

Um ponteiro também tem tipo. Ex: Int, Float, Char, etc.

Ponteiro

Para declarar um ponteiro temos a seguinte forma geral:

```
tipo_do_ponteiro *nome_da_variável;
```

É o asterisco (*) que faz o compilador saber que aquela variável não vai guardar um valor mas sim um endereço para aquele tipo especificado.

Ponteiro

```
int *pt;  
char *temp, *pt2;
```

Essas variáveis ainda não foram inicializados. Isto significa que eles apontam para um lugar indefinido.

O ponteiro deve ser inicializado (apontado para algum lugar conhecido) antes de ser usado! Isto é de suma importância!

Ponteiro

Para atribuir um ponteiro a um endereço de uma variável basta usar o operador &. Veja o exemplo:

```
int count=10;  
int *pt;  
pt=&count; // atribui o ponteiro *pt ao  
            endereço count e consequentemente  
            recebe o valor 10
```

Ponteiro

```
int count=10;
```


```
int *pt;
```

```
pt=&count;
```

Criamos um inteiro count com o valor 10 e um apontador para um inteiro pt. A expressão &count nos dá o endereço de count, o qual armazenamos em pt. Podemos, por exemplo, alterar o valor de count usando pt. Usando o operador *. Ex: *pt=12;


```
#include <stdio.h>
int main ()
{
    int num,valor;
    int *p;
    num=55;
    p=&num;
    /* Pega o endereco de num */
    valor=*p;
    /* Valor e igualado a num de uma
    maneira indireta */
    printf ("\n\n%d\n",valor);
    printf ("Endereco para onde o ponteiro aponta: %p\n",p);
    printf ("Valor da variável apontada: %d\n",*p);
    return(0);
}
```

EXEMPLO 1



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

```
int main ()
```

```
{
```

```
int num,*p;
```

```
num=55;
```

```
p=&num;
```

```
/* Pega o endereco de num */
```

```
printf ("\nValor inicial: %d\n",num);
```

```
*p=100; /* Muda o valor de num de uma maneira indireta*/
```

```
printf ("\nValor final: %d\n",num);
```

```
return(0);
```

```
}
```

EXEMPLO 2

Operações com Ponteiro

`p1=p2;`

p1 aponte para o mesmo lugar que p2.

`*p1=*p2;`

p1 tenha o mesmo conteúdo da variável apontada por p2.

`P++;` ou `p--;`

Incrementa ou decrementa o endereço de memória, depende do tipo. se for incremento de um ponteiro `char*` ele anda 1 byte, se for `double*` ele anda 8 bytes na memória.

Operações com Ponteiro

`(*p) ++;`

Incrementa o conteúdo da variável apontada pelo ponteiro p.

`==` ou `!=`

Saber se dois ponteiros são iguais ou diferentes.

`>`, `<`, `>=` e `<=`

Estamos comparando qual ponteiro aponta para uma posição mais alta na memória.



Operações com Ponteiro

Há entretanto operações que você não pode efetuar num ponteiro. Você não pode dividir ou multiplicar ponteiros, adicionar dois ponteiros, adicionar ou subtrair floats ou doubles de ponteiros.

Exercício

```
int main()
{
    int y, *p, x;
    y = 0;
    p = &y;
    x = *p;
    x = 4;
    (*p)++;
    x--;
    (*p) += x;
    printf ("y = %d\n", y);
    return(0);
}
```

Qual o valor de y no final do programa?

Ponteiros com Vetor/Matriz

Considere o seguinte algoritmo que zera uma matriz de 50 linhas e 50 colunas

```
int main ()
{
float matrx [50][50];
int i,j;
for (i=0;i<50;i++)
for (j=0;j<50;j++)
matrx[i][j]=0.0;
return(0);
}
```

Podemos reescrever o algoritmo anterior usando ponteiro:

```
int main ()
{
float matrx [50][50];
float *p;
int count;
p=matrx[0];
for (count=0;count<2500;count++)
{
*p=0.0;
p++;
}
return(0);
}
```


Exemplo de ponteiro com string

```
#include <stdio.h>

void StrCpy (char *destino, char *origem)
{
    while (*origem)
    {
        *destino=*origem;
        origem++;
        destino++;
    }
    *destino='\0';
}
```

Exemplo de ponteiro com string

```
int main ()
{
char str1[100],str2[100],str3[100];
printf ("Entre com uma string: ");
gets (str1);
StrCpy (str2,str1);
StrCpy (str3,"Voce digitou a string ");
printf ("\n\n%s%s",str3,str2);
return(0);
}
```



Exercício 1

Escreva um programa que declare uma matriz 100x100 de inteiros.

Deverá inicializar a matriz com zeros, usando ponteiros para endereçar os elementos. Preencha depois a matriz com números de 1 a 10000, também usando ponteiros.

Referências

FORBELLONE, André Luiz Villar; EBERSPACHER, Henri Frederico. **Lógica de programação**: a construção de algoritmos e estruturas de dados. 3. ed. São Paulo: Prentice Hall, 2005.