2º Semestre

Técnicas de Programação

Luiz Fernando Carvalho

luizfcarvalhoo@gmail.com

Alocação dinâmica de memória

- Quando você declara um vetor em um programa em C, você deve informar quantos elementos devem ser reservados
  - Se você conhece esse número a priori, é trivial.
- Caso o tamanho do vetor não seja conhecido, deve-se definir um tamanho máximo para acomodar os dados
  - **Desperdício de memória** caso poucos valores forem armazenados no vetor;
  - **Falta de memória** caso o vetor declarado seja insuficiente para armazenar os dados;
- A solução é usar ALOCAÇÃO DINÂMICA;

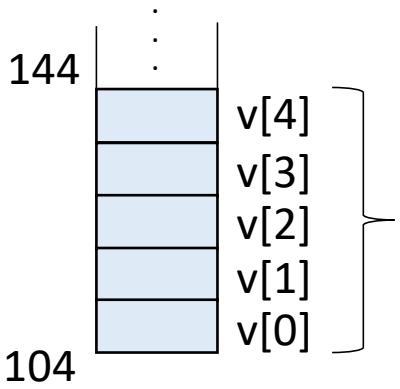
Alocação dinâmica de memória

- **Alocação Dinâmica** é o processo de solicitar e utilizar memória durante a execução de um programa;
- Aplicada para que um programa utilize apenas a memória necessária para sua execução, sem desperdício de memória;
- Sendo assim, alocação dinâmica de memória deve ser utilizada quando não se sabe, por algum motivo, quanto espaço de memória será necessário para o armazenamento de algum ou alguns valores;

Alocação estática de memória

- Dando uma olhada na **alocação estática** de memória...
- Quando declaramos um vetor v com 5 posições para armazenar inteiros, o compilar reserva na pilha, 5 espaços de memória para armazenar esses valores inteiros;
- Os valores são armazenados sequencialmente, formando um bloco **contínuo**
  - Por conta disso, podem ser acessados por meio de um índice!

Se cada **int** ocupa 4 bytes, então: 4 bytes * 5 = 20 bytes contínuos na pilha



Lembre-se que v[5] não existe. Estaria sobrescrevendo o valor de uma outra variável armazenada antes (ou depois) do vetor

Alocação estática de memória

- A alocação estática é feita em “tempo de compilação”;
  - Todo espaço de memória usado pelo programa é definido durante a compilação;
  - Nenhum espaço extra para as variáveis pode ser requerido durante a execução;

```
1  int typedef struct{
2       char nome[30];
3       int idade;
4  }Pessoa;
5  int main(){
6       int n;
7       char a;
8       float num;
9       int v[5];
10      Pessoa p;
11
12      return 0;
13 }
```

Variáveis alocadas estaticamente

Alocação dinâmica de memória

- A alocação dinâmica é feita em “tempo de execução”;
  - Durante a execução do programa, mais ou menos memória pode ser utilizada baseada na demanda da aplicação;

- No padrão ANSI C, existem 4 funções para se utilizar na alocação de memória:
  - `malloc;`
  - `calloc;`
  - `realloc;`
  - `free;`

- Todas essas funções pertencem à biblioteca `stdlib.h`

malloc (*memory allocation*)

```
void *malloc(num_bytes);
```

- Essa função recebe como parâmetro `num_bytes` correspondente a quantidade de bytes consecutivos que se deseja alocar.
- O retorno é um ponteiro `void`, podendo ser atribuído a qualquer tipo de ponteiro;
  - O ponteiro indica a posição na memória em que se inicia o bloco de memória alocada;
  - Caso o retorno seja NULL, a memória não pôde ser alocada;

```
1  int main(){
2       char *ptr;
3       ptr = malloc(1); //aloca 1 byte, ptr aponta para esse byte
4
5       return 0;
6  }
```

Relembrando ponteiros

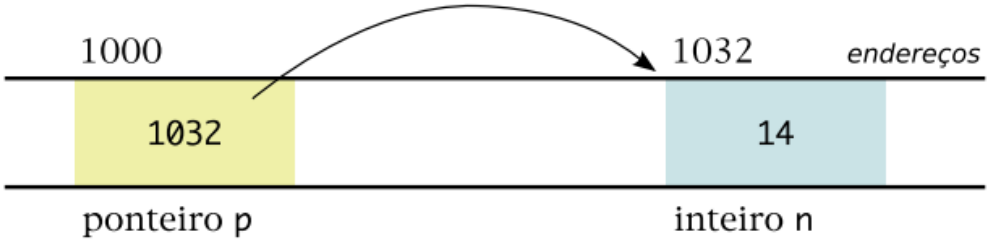
- Considerando uma variável declarada como:

```
int* p;
```

- ***p*** é um ponteiro para **int**, isto é, uma variável que armazena o endereço de uma variável do tipo **int**.
- Supondo que ***p*** armazene o valor 1032, tem-se que:

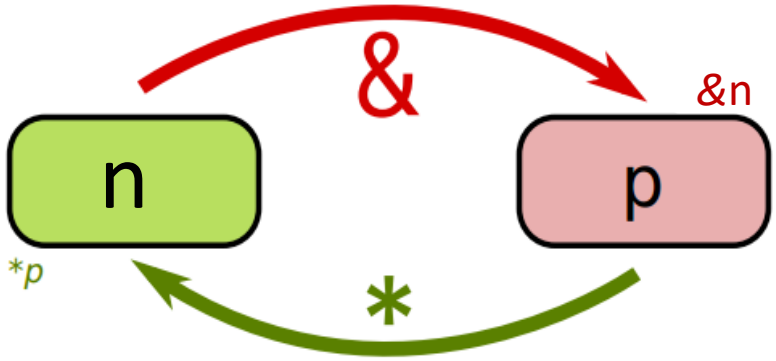
```
p = &n;
```

- Define-se **p*** como sendo o valor contido na posição de memória apontada por ***p***. Assim, **p*** vale 14.



Relembrando ponteiros

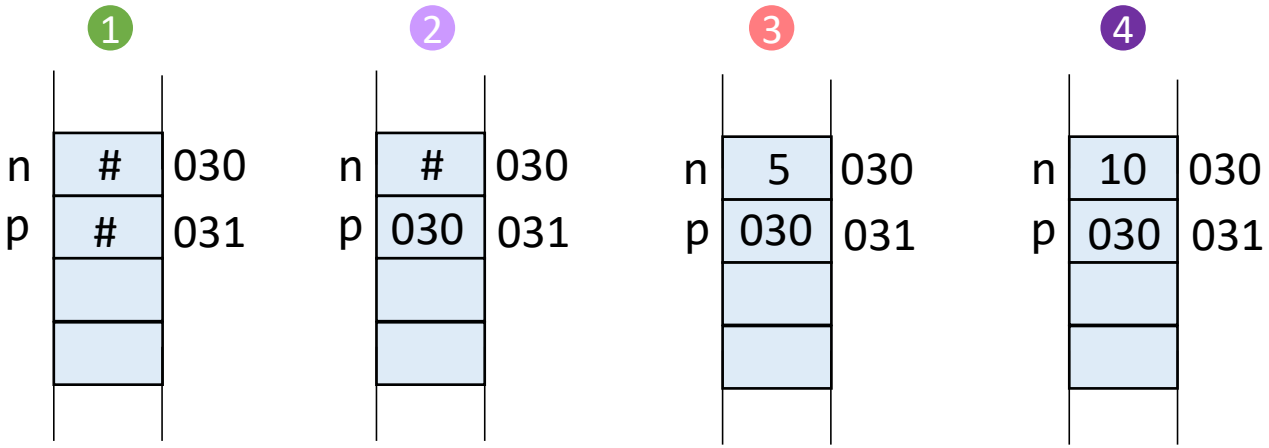
- Para acessar a variável que é apontada por um ponteiro, usamos o operador * (o mesmo asterisco usado na declaração);
- Se **p** é um ponteiro, podemos acessar a variável para a qual ele aponta com ***p**.
  - Essa expressão pode ser usada tanto para ler o conteúdo da variável quanto para alterá-lo.



Relembrando Ponteiros

- Acessando indiretamente valores das variáveis por ponteiros

```
1  int main(){
2    1 int n, *p;
3    2 p = &n; //p aponta para a variável n
4
5    3 *p = 5;
6      printf("n = %d", n); //imprime 5
7    4 n = 10;
8      printf("*p = %d", *p); //imprime 10
9
10     return 0;
11 }
```



malloc (*memory allocation*)

```
void *malloc(num_bytes);
```

- Se precisarmos alocar memória para uma estrutura complexa, pode-se usar o operador `sizeof`, que diz quantos bytes tem a estrutura;

```
1  typedef struct{
2       int dia, mes, ano;
3  }Data;
4
5  int main(){
6       Data *d;
7       d = malloc(sizeof(Data)); //aloca memoria para armazenar uma variável Data
8       ...
9
10      return 0;
11 }
```

malloc (*memory allocation*)

- Como falamos anteriormente, o ponteiro de retorno da função `malloc` é genérico:
  - Ponteiro é convertido automaticamente para o tipo apropriado;
  - O ponteiro pode ser convertido explicitamente se o programador assim desejar;

```
1  int main(){
2     int *vet;
3
4     vet = malloc(10*sizeof(int));
5     ...
6
7     return 0;
8  }
```

Mais comum de ser encontrado

```
1  int main(){
2     int *vet;
3
4     vet = (int *)malloc(10*sizeof(int));
5     ...
6
7     return 0;
8  }
```

São equivalentes!

Free

```
void free(void* ptr);
```

- As variáveis alocadas estaticamente dentro de uma função (variáveis locais), desaparecem assim que a execução da função termina;
- Variáveis alocadas dinamicamente continuam a existir mesmo depois que a execução da função termina;
- A função `free` desaloca a porção de memória alocada;
- Recebe como parâmetro o ponteiro para a região de memória a ser desalocada;
  - Se o ponteiro não apontar para uma região previamente alocada, a função tem um comportamento indefinido;
  - Se o ponteiro estiver definido como NULL, a função não faz nada;

Free

```
void free(void* ptr);
```

```
1  int main(){
2     int *vet;
3
4     vet = (int *)malloc(10*sizeof(int));
5     ...
6     free(vet);
7
8     return 0;
9  }
```

Exemplo

Alocando um vetor de inteiros dinamicamente e calculando a soma de seus elementos.

```
1  int main(){
2     int *vet, tam, i, soma=0;
3
4     printf("Informe o tamanho do vetor: ");
5     scanf("%d", &tam);
6
7     vet = (int *)malloc(tam*sizeof(int));
8     for(i=0;i<tam;i++){
9         printf("informe o vet[%d]: ", i);
10        scanf("%d", &vet[i]);
11    }
12    for(i=0;i<tam;i++)
13       soma += vet[i];
14
15    printf("A soma dos elementos do vetor e' %d", soma);
16    free(vet);
17
18    return 0;
19 }
```

Alocando memória para um vetor do tamanho escolhido pelo usuário

Desalocando memória usada pelo vetor