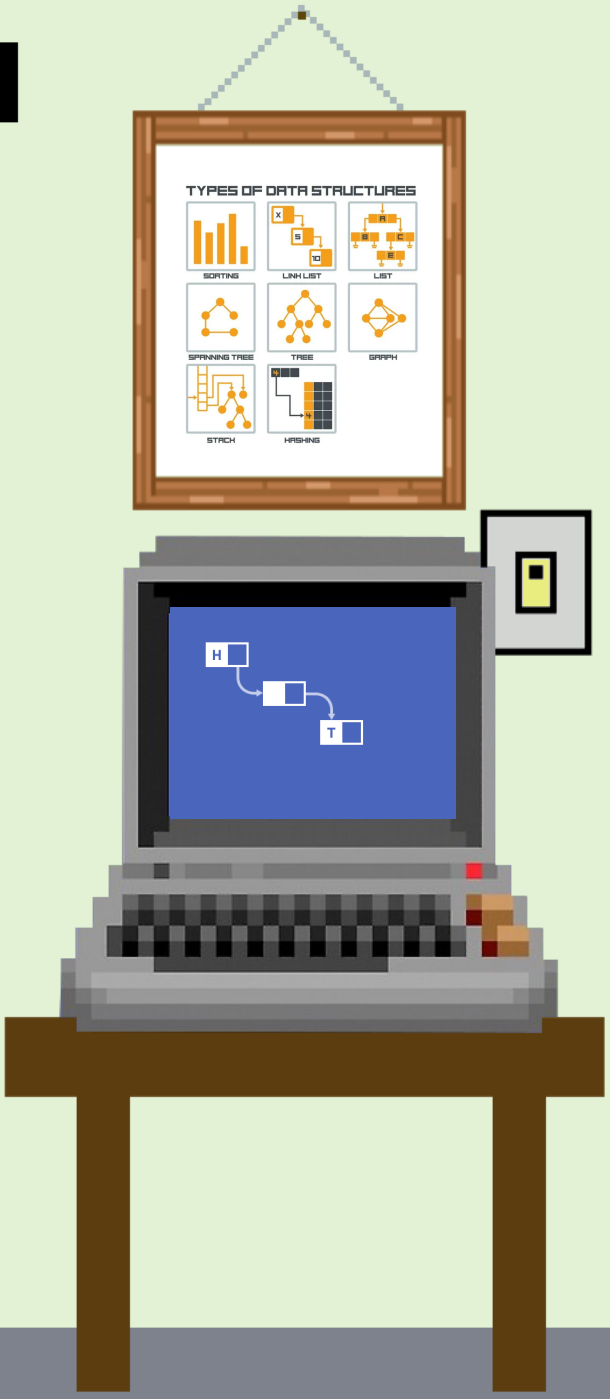
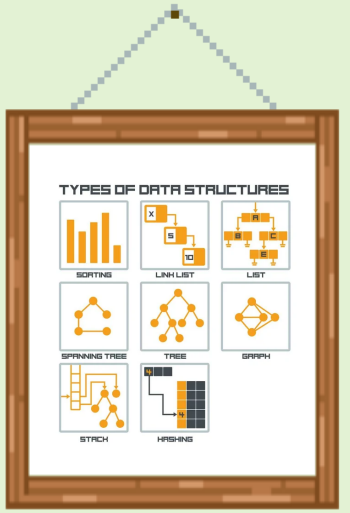
Gerenciamento de Memória

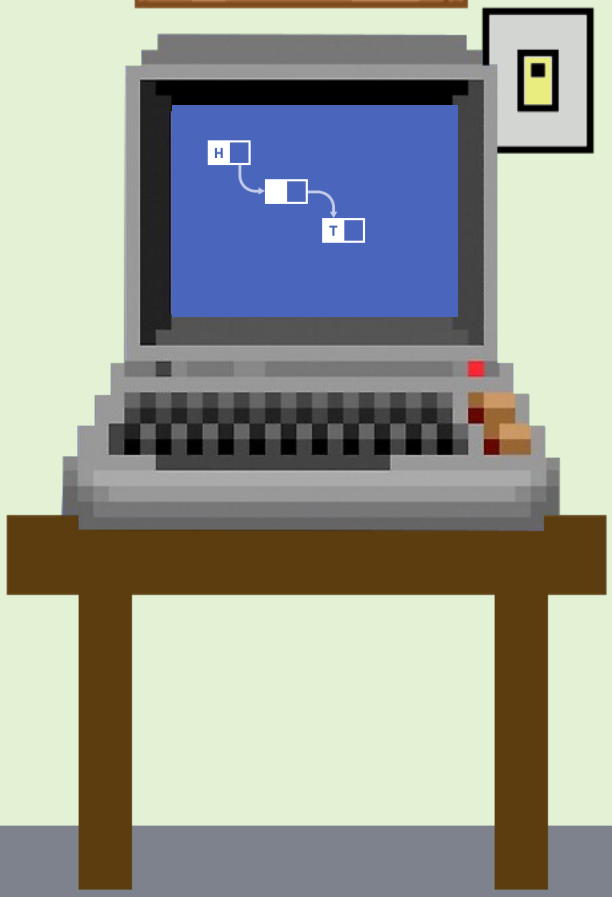
Estruturas de Dados I



Memória em Execução

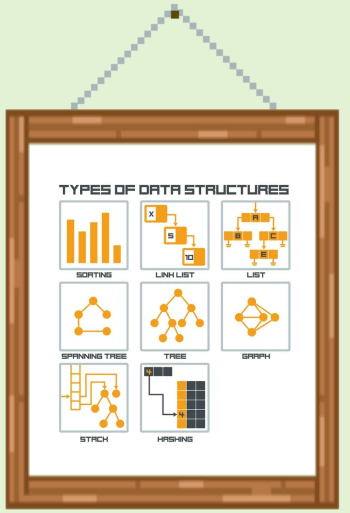
- Maneiras de reservar o espaço da memória
  - Variáveis globais (estáticas)
    - Espaço ocupado na memória durante toda a execução do programa
  - Variáveis locais (stack)
    - Espaço ocupado enquanto o bloco onde a variável foi declarada estiver executando
  - Espaços dinâmicos (heap)
    - Espaço ocupado até ser explicitamente liberado

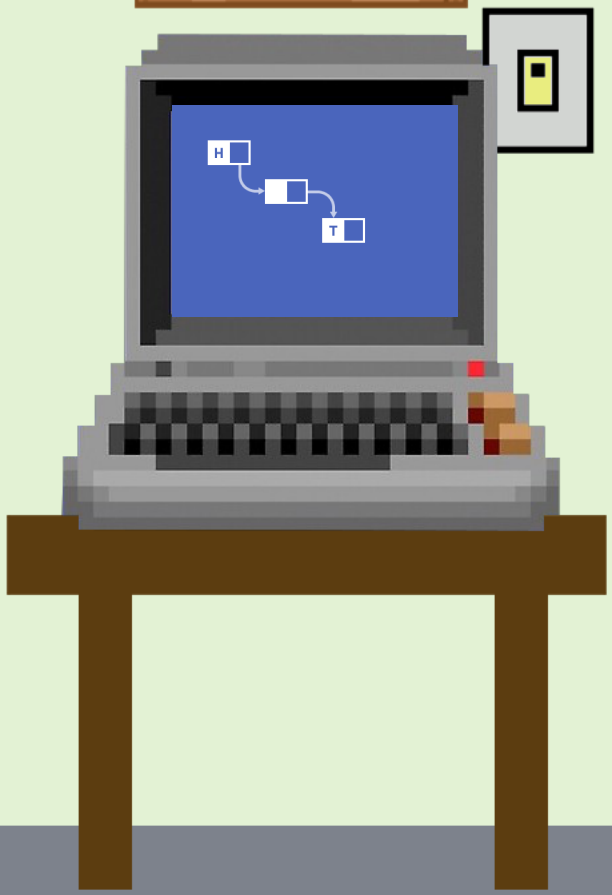




Alocação de Memória

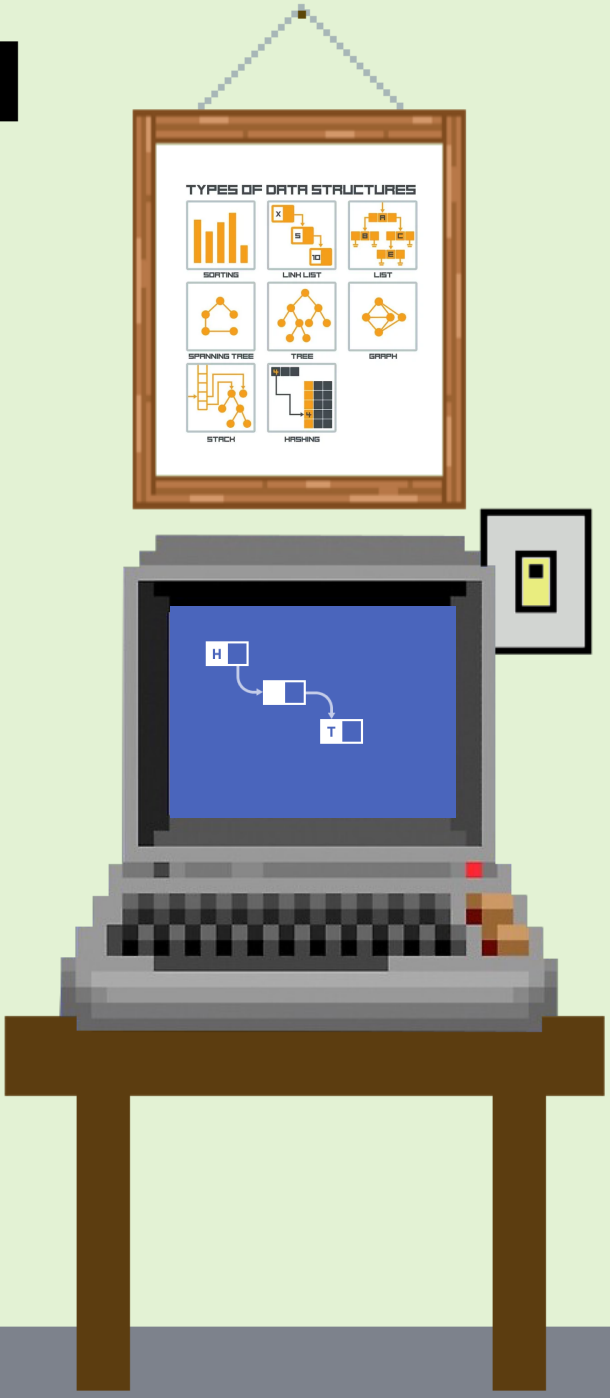
- Pode ser estática ou dinâmica
- Estática
  - Ocorre antes que blocos do programa executem
  - São excluídas assim que os blocos encerram
  - Não possui relação com a palavra-chave static
- Dinâmica
  - Realizada ao longo da execução do programa
  - Útil quando a quantidade de memória a alocar só é conhecida durante a execução do programa
  - Funções malloc, realloc e free (stdlib)





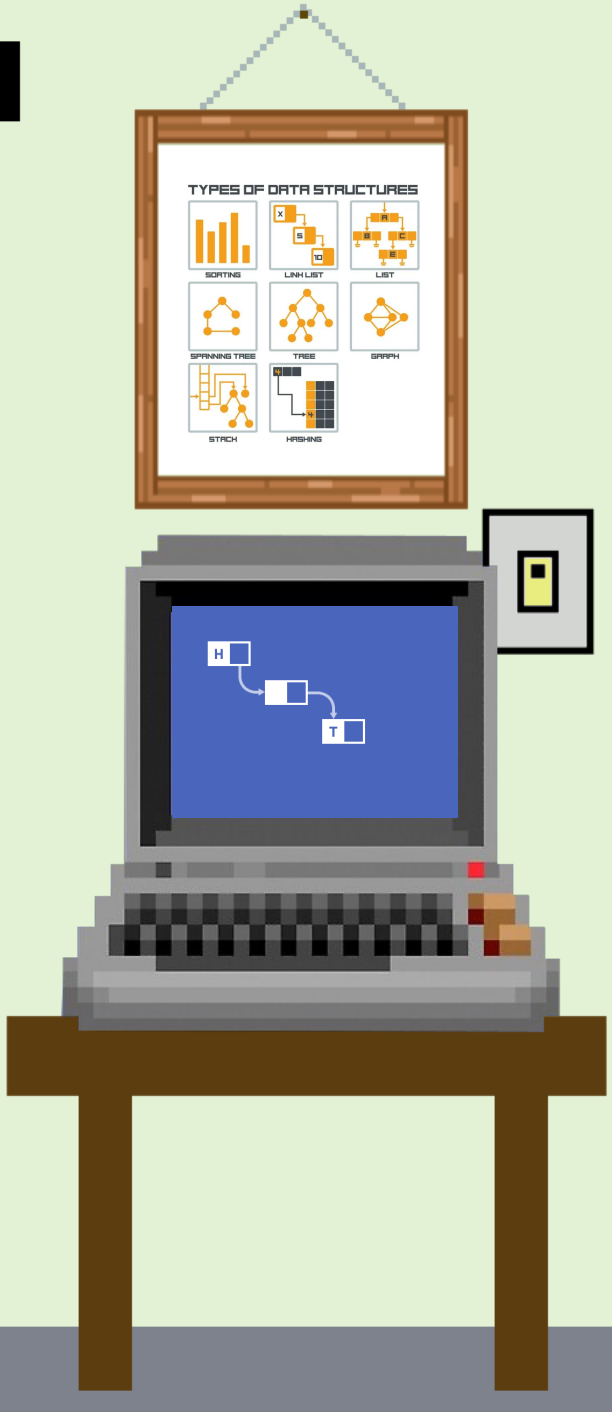
Alocação Estática

- Toda a memória a ser utilizada é alocada de uma vez
  - Independente da quantidade usada na execução
- O espaço máximo depende do hardware
  - Tamanho da memória "endereçável"
- Exemplo
  - int v[1000]
    - Espaço contíguo na memória para 1000 inteiros
    - Cada int ocupa 4 bytes (total ~4KB)
  - char v[1000]
    - Espaço contíguo na memória para 1000 chars
    - Cada char ocupa 1 byte (total ~1KB)



Alocação Dinâmica

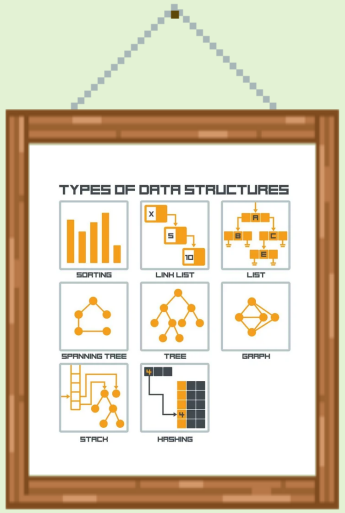
- Aloca a memória sob demanda
  - Aloca, libera e realoca durante a execução
- O SO seleciona blocos de memória que estão livres
  - A memória alocada pode conter dados anteriores
  - É necessário inicializar
- É responsabilidade do programador liberar a memória
- Exemplo
  - char *c = (char *) malloc(1000)
    - Espaço na memória de tamanho 1000 bytes
  - int *v = (int *) malloc(1000 * sizeof(int))
    - Espaço na memória de tamanho 4000 bytes

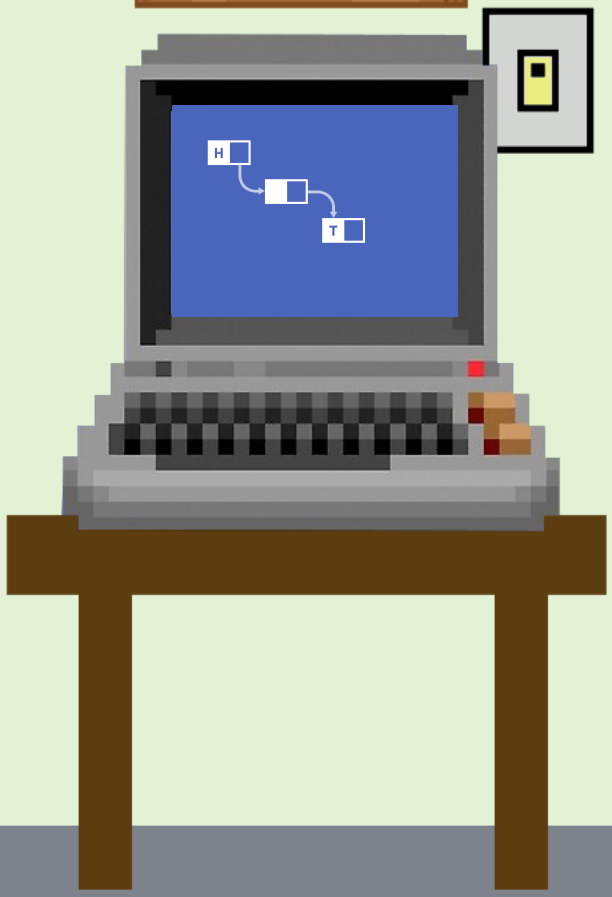


Alocação Dinâmica

- malloc
  - Abreviatura de memory allocation
  - Aloca um bloco de bytes consecutivos na memória e devolve o endereço para o início desse bloco
  - O número de bytes é passado como argumento
  - O programador deve armazenar o endereço retornado em um ponteiro de tipo apropriado

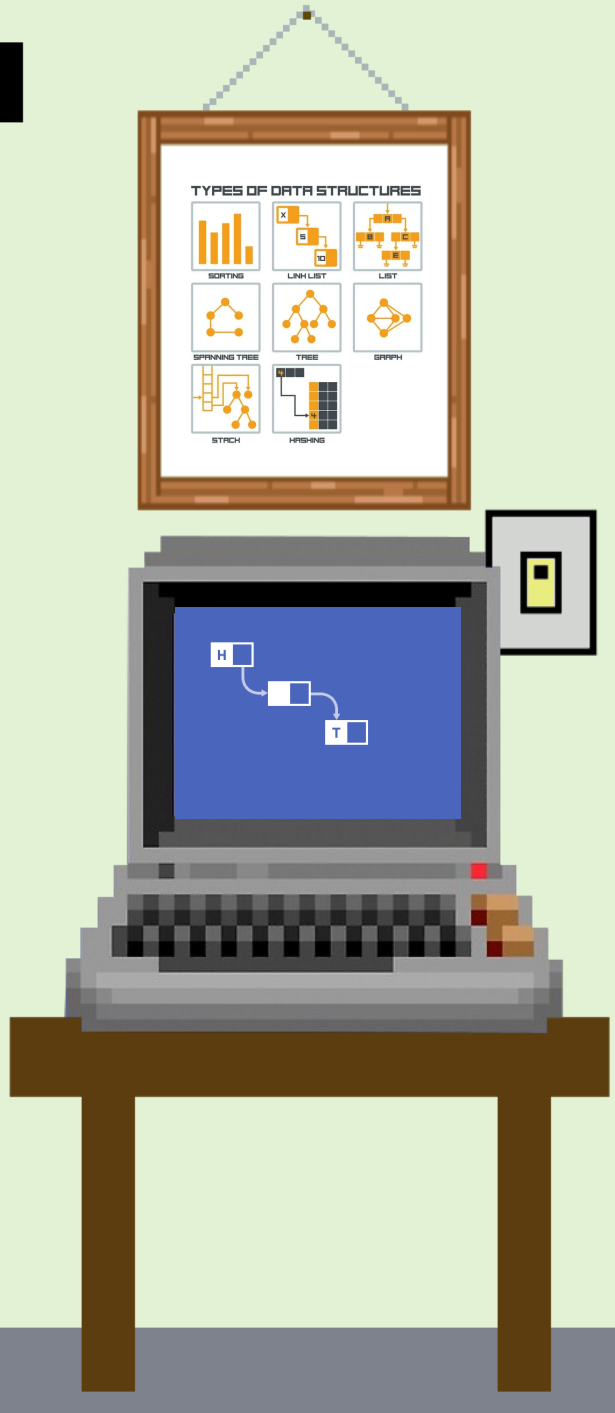
```
char *ptr;
ptr = malloc (1);
scanf ("%c", ptr);
```





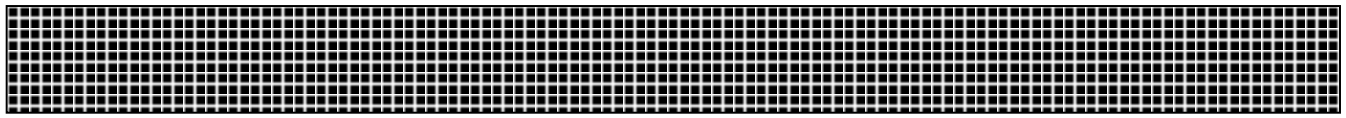
Alocação Estática vs Dinâmica

- Alocação Estática
  - Alocação fixa de memória pode ser ineficiente
  - Tamanhos fixos não são comuns a tudo
  - Comum ter espaços alocados e não utilizados
- Alocação Dinâmica
  - O programa pode solicitar e devolver memória enquanto está em execução
  - Requer gerenciamento pelo desenvolvedor

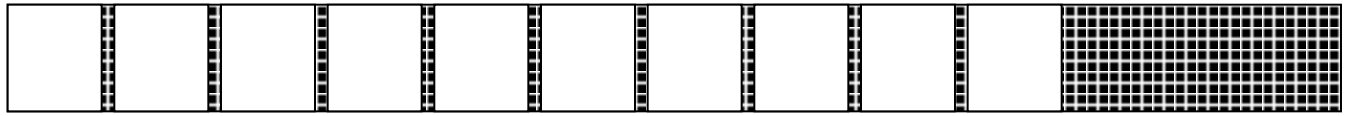


Alocação Estática vs Dinâmica

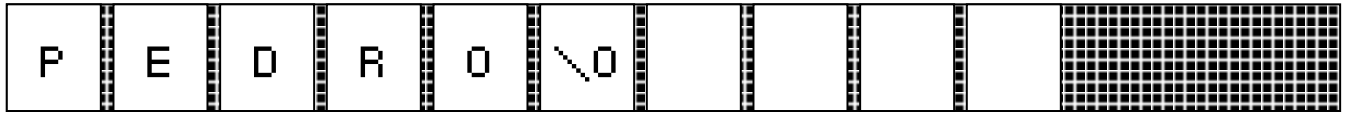
- Ex: Alocar espaço para o nome de uma pessoa
  - Vetor de string (alocação estática)
    - Tamanho máximo do nome (não ideal)
    - A maioria dos nomes não tem o mesmo tamanho

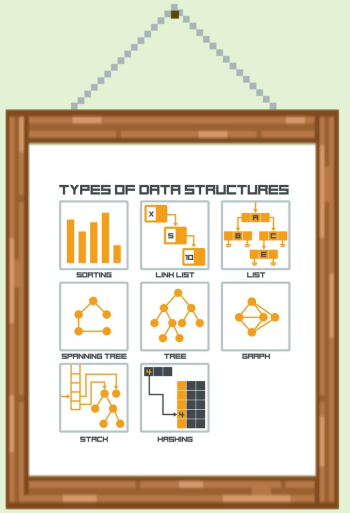


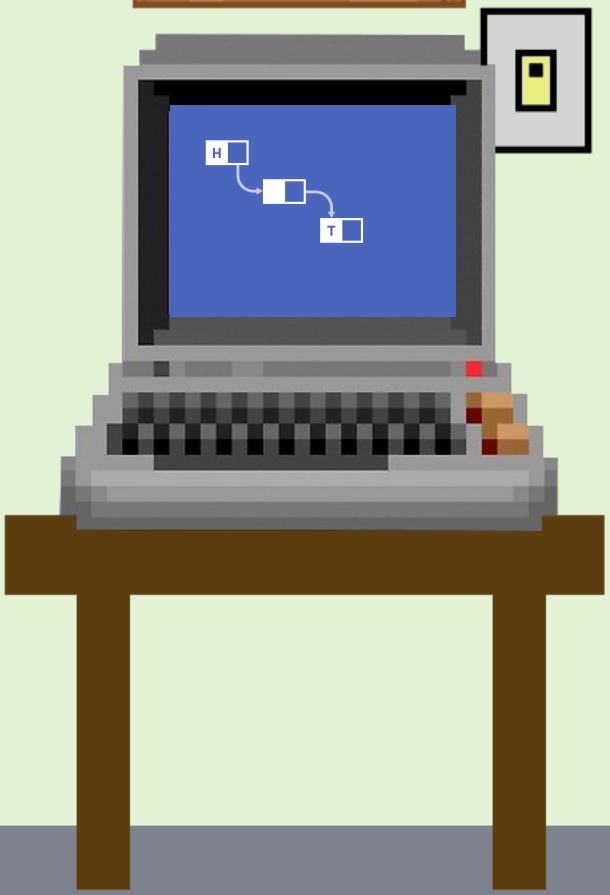
int nome[10];



scanf("%s", nome);

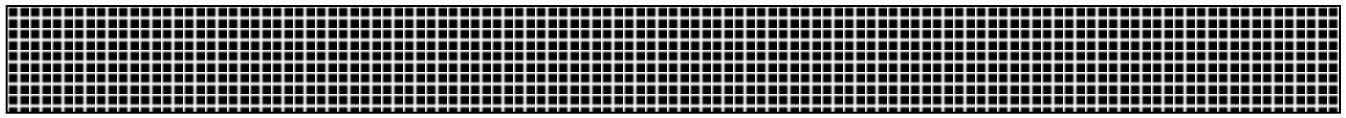




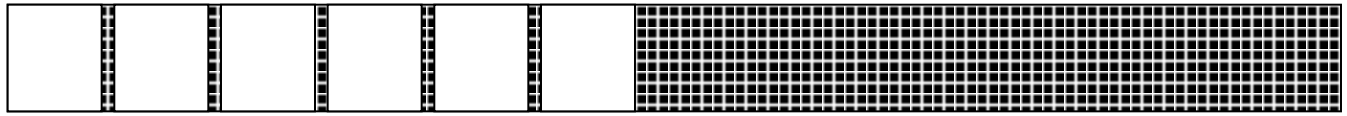


Alocação Estática vs Dinâmica

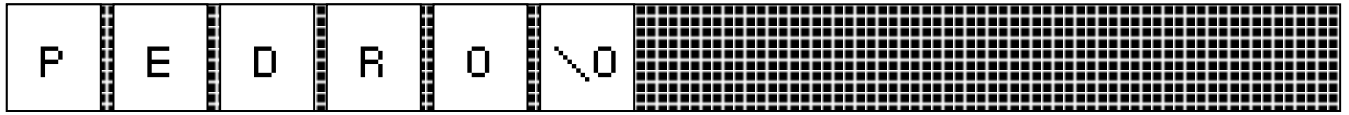
- Ex: Alocar espaço para o nome de uma pessoa
  - Vetor de string (alocação dinâmica)
    - Tamanho do nome conforme a necessidade
    - O programador define quando e quanto alocar

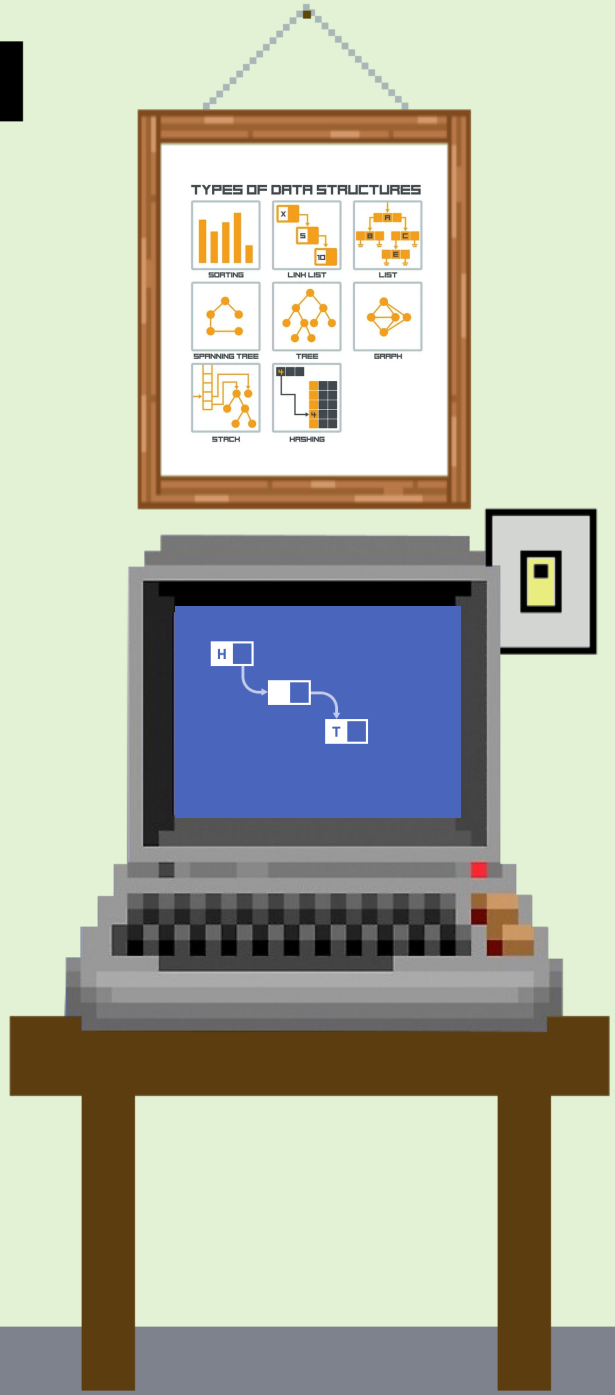


```
int* nome = (int *) malloc(6);
```



```
scanf("%s", nome);
```

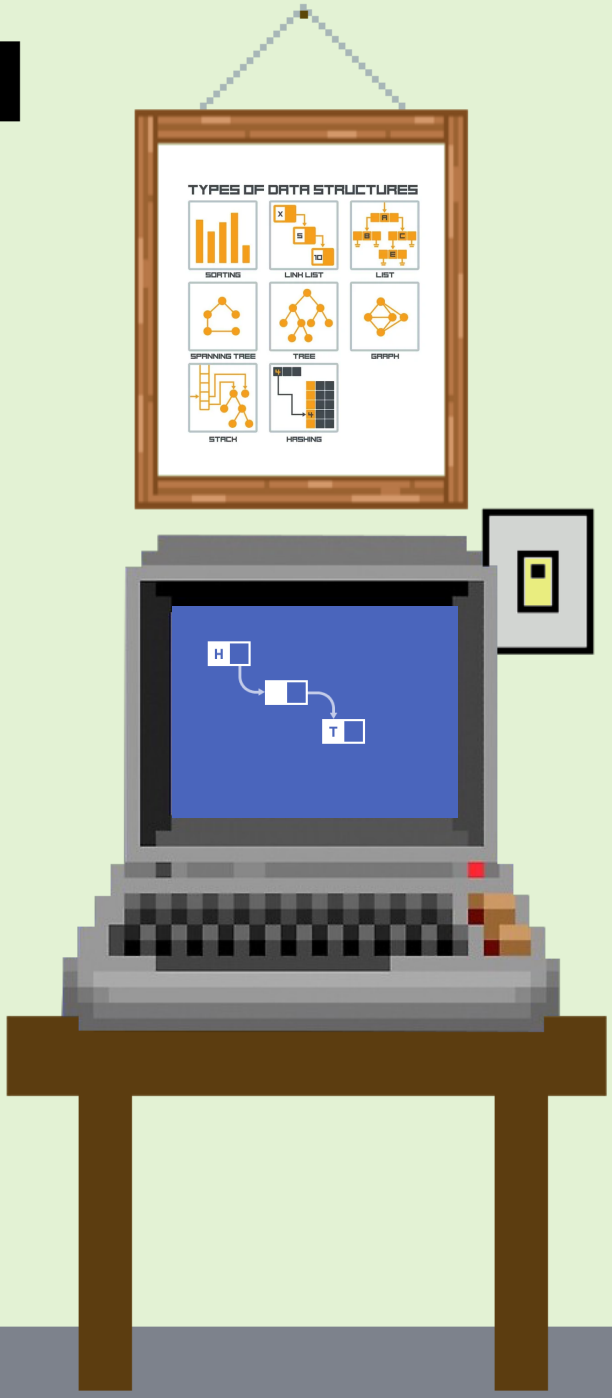




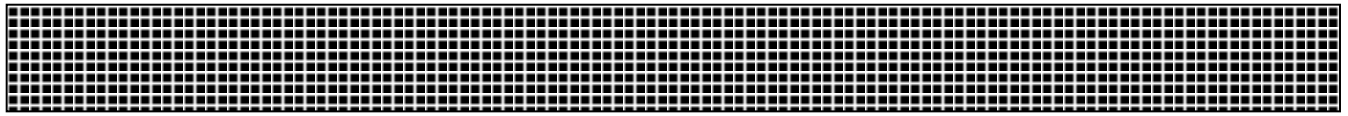
Liberação de Memória

- free
  - Variáveis alocadas dinamicamente continuam a existir mesmo depois que blocos são executados
  - Libera a memória alocada dinamicamente
  - Avisa ao sistema que o bloco de bytes apontado por um ponteiro está disponível para reciclagem

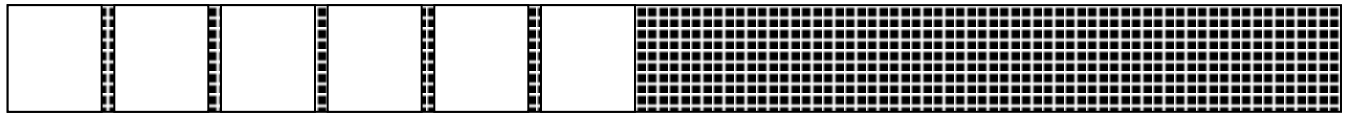
```
char *ptr;
ptr = malloc (1);
free(ptr);
```



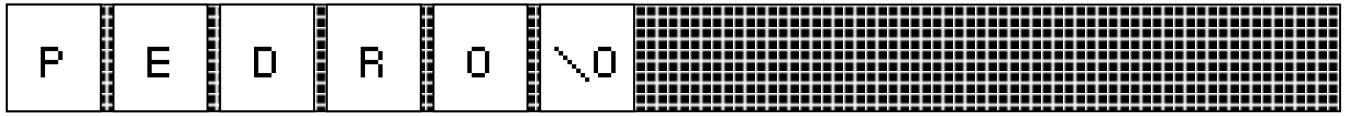
Alocação Dinâmica de Memória



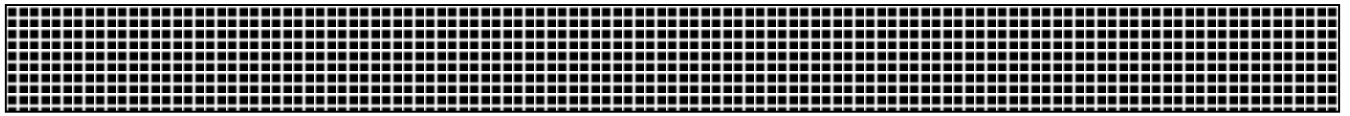
```
int* nome = (int *) malloc(6);
```

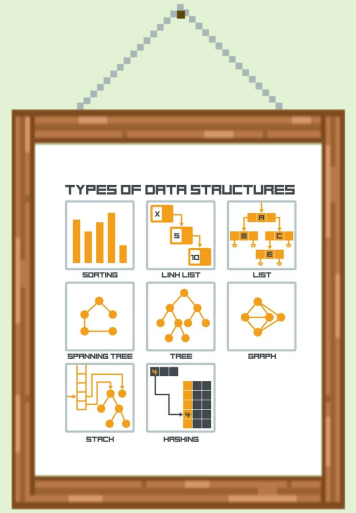


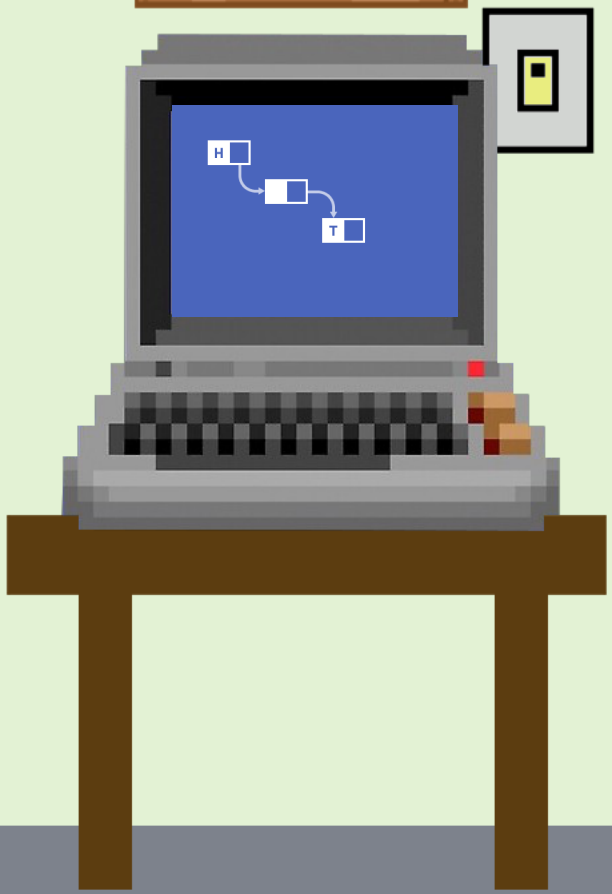
```
scanf("%s", nome);
```




```
free(nome)
```



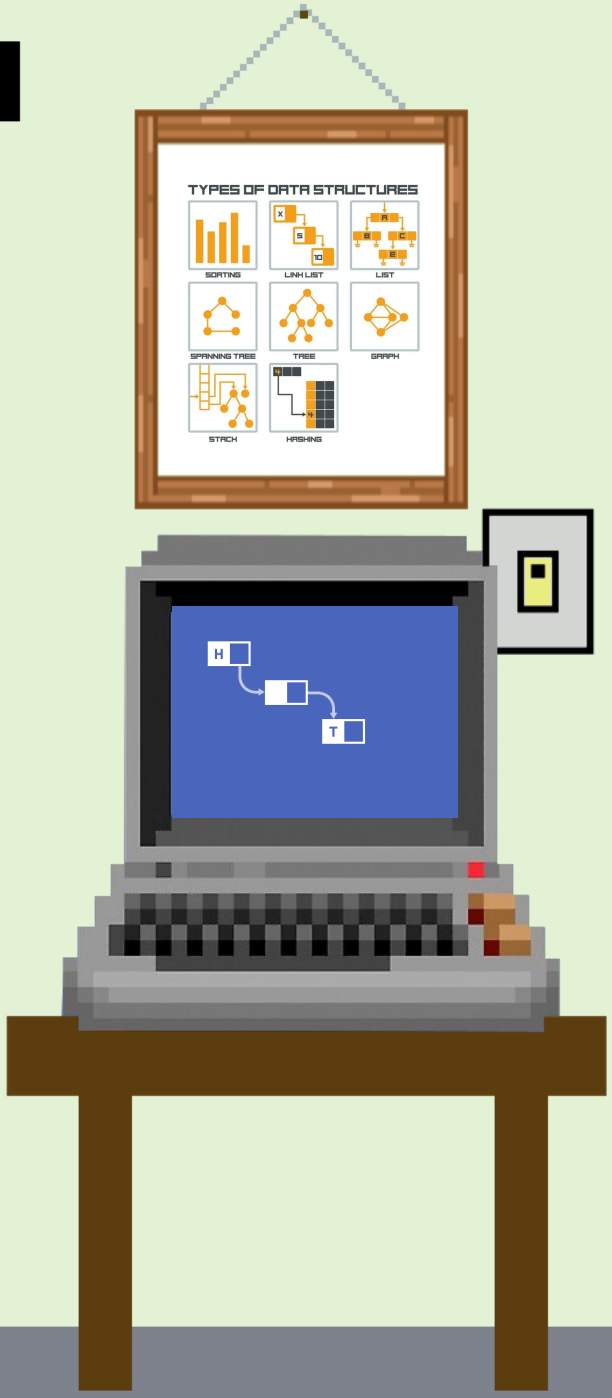




Alocação Dinâmica de Memória

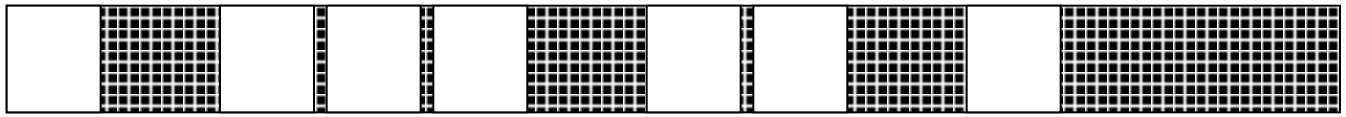
- realloc
  - Altera durante a execução do programa, o tamanho de um bloco de bytes que foi alocado por malloc
  - Recebe o endereço de um bloco previamente alocado e o novo tamanho que o bloco deve ter
  - Aloca um novo bloco, copia o conteúdo do bloco original, e devolve o endereço do novo bloco

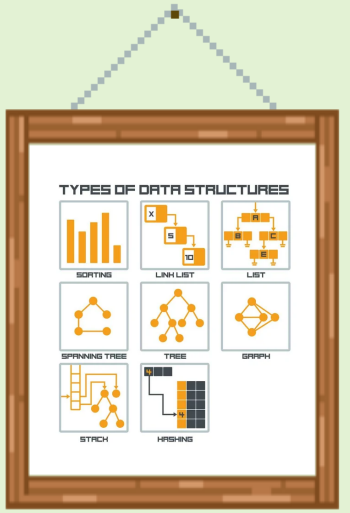
```
int *v = malloc (1000 * sizeof (int));
v = realloc (v, 2000 * sizeof (int));
```

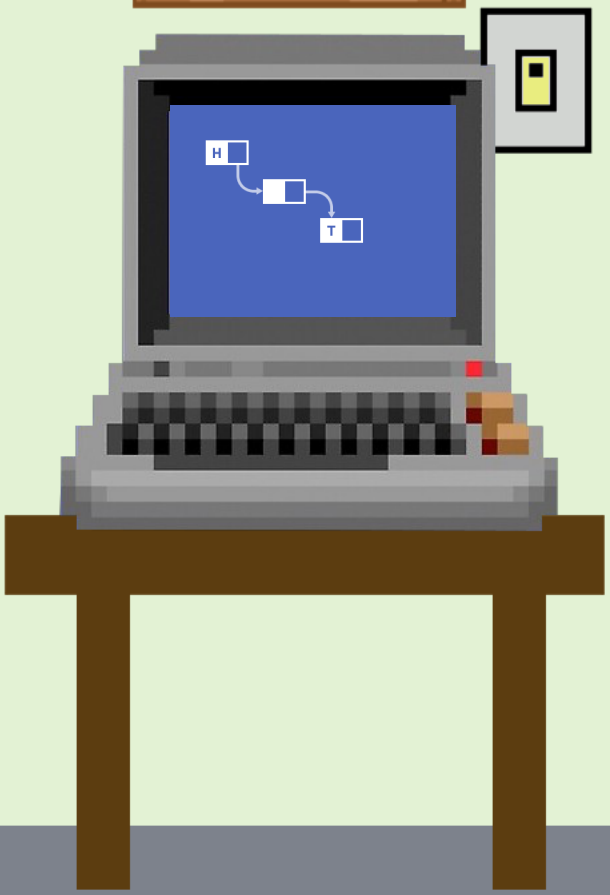


Desvantagens da Alocação Dinâmica

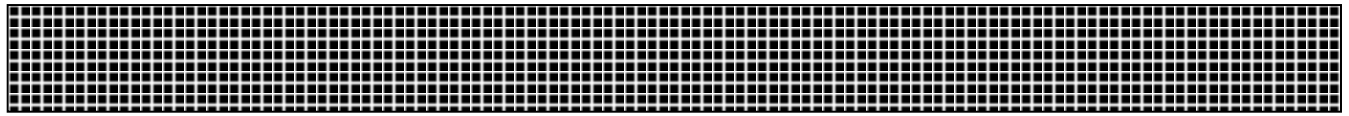
- O programador deve gerenciar a memória
  - Invasão de memória estranha (memory violation)
  - Não devolver memória já utilizada
- Fragmentação
  - Blocos livres de memória não contíguos
  - Estruturas encadeadas fazem melhor uso da memória fragmentada



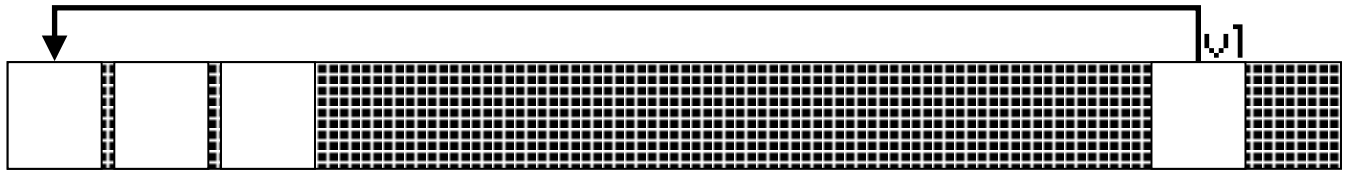


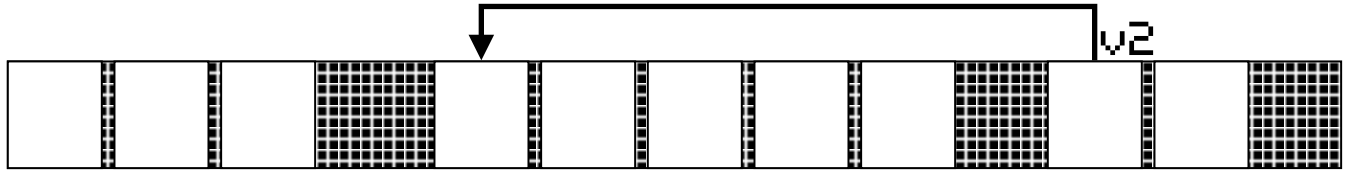


Alocação Dinâmica e Ponteiros

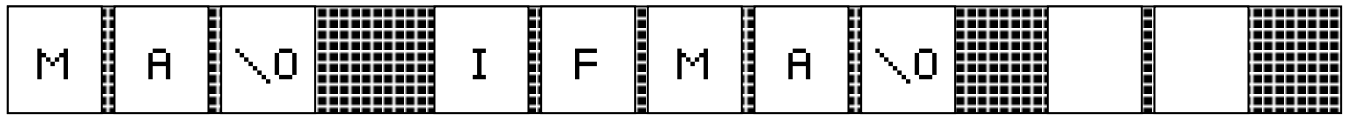


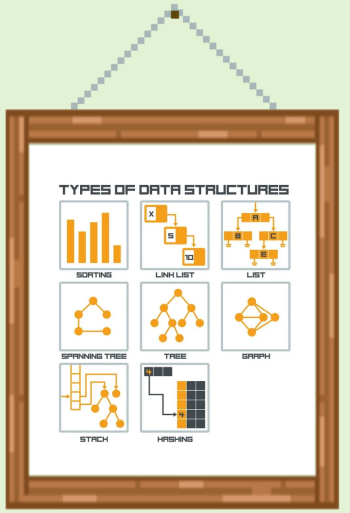
```
int* v1 = (int *) malloc(3);
int* v2 = (int *) malloc(5);
```

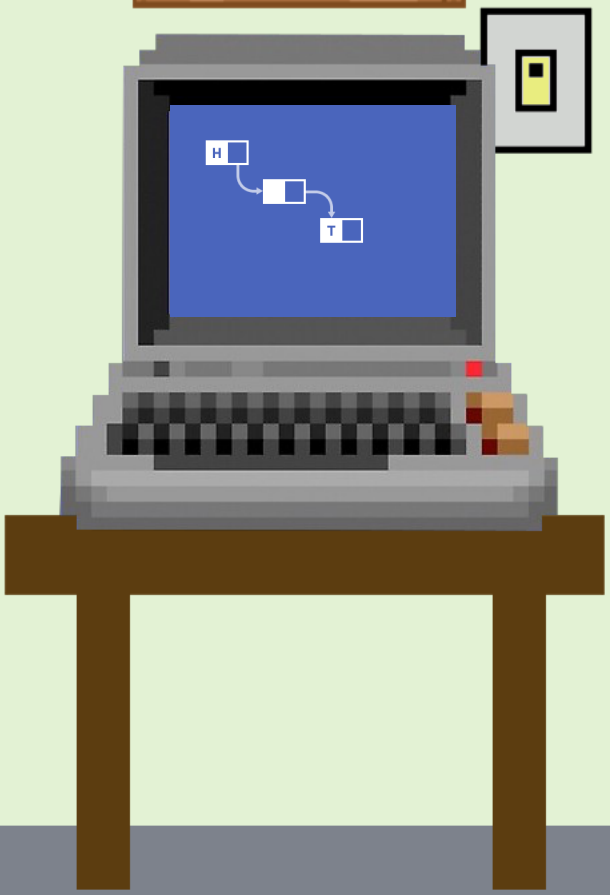




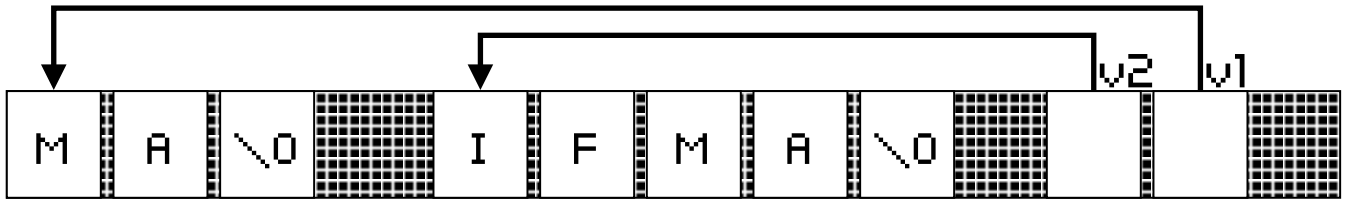

```
scanf("%s", v1);
scanf("%s", v2);
```



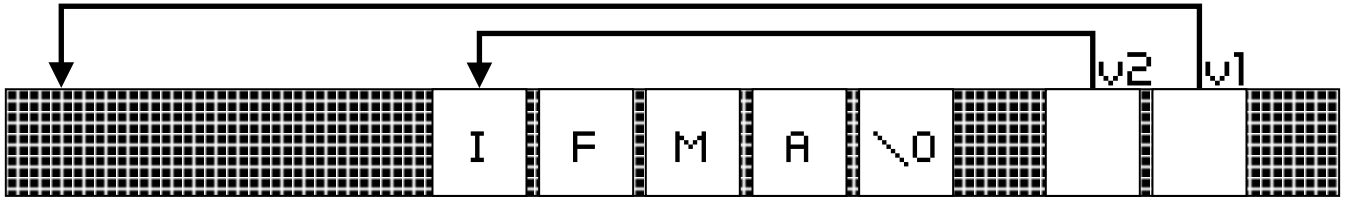




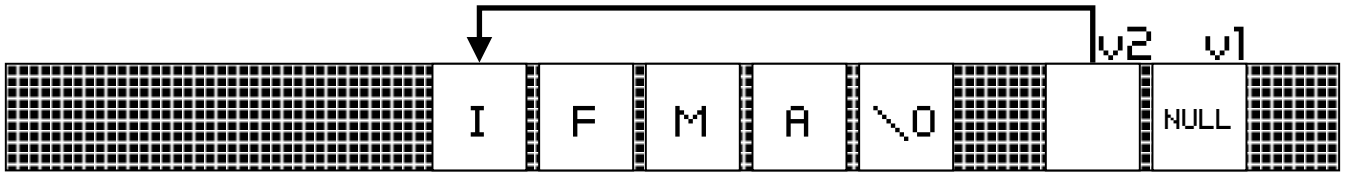
Alocação Dinâmica e Ponteiros

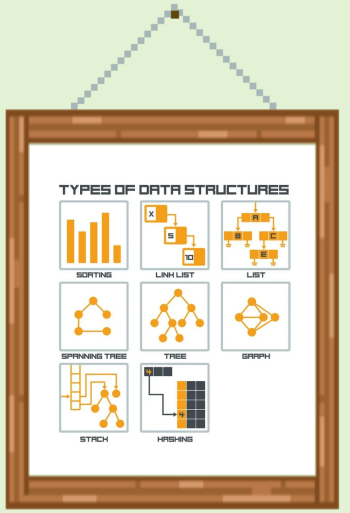


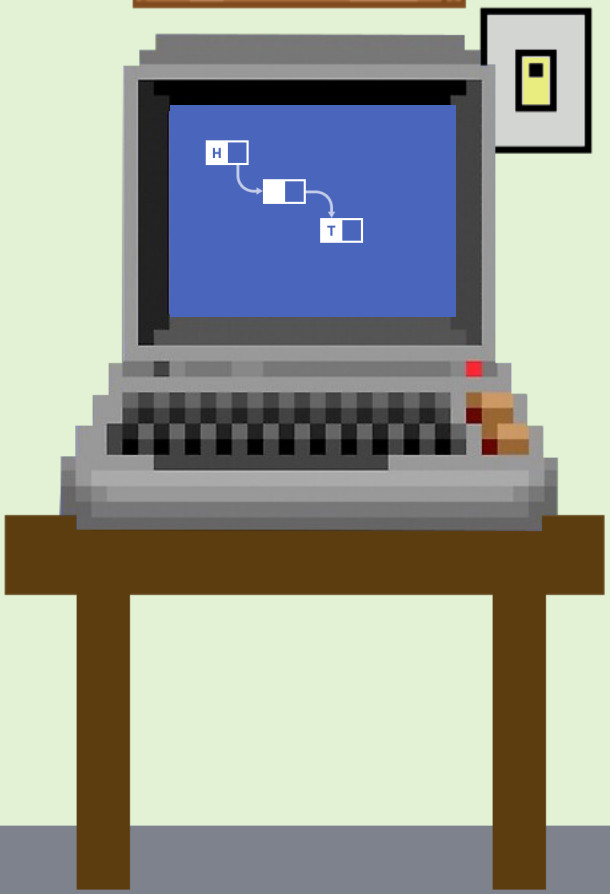
free(v1);



v1=NULL;

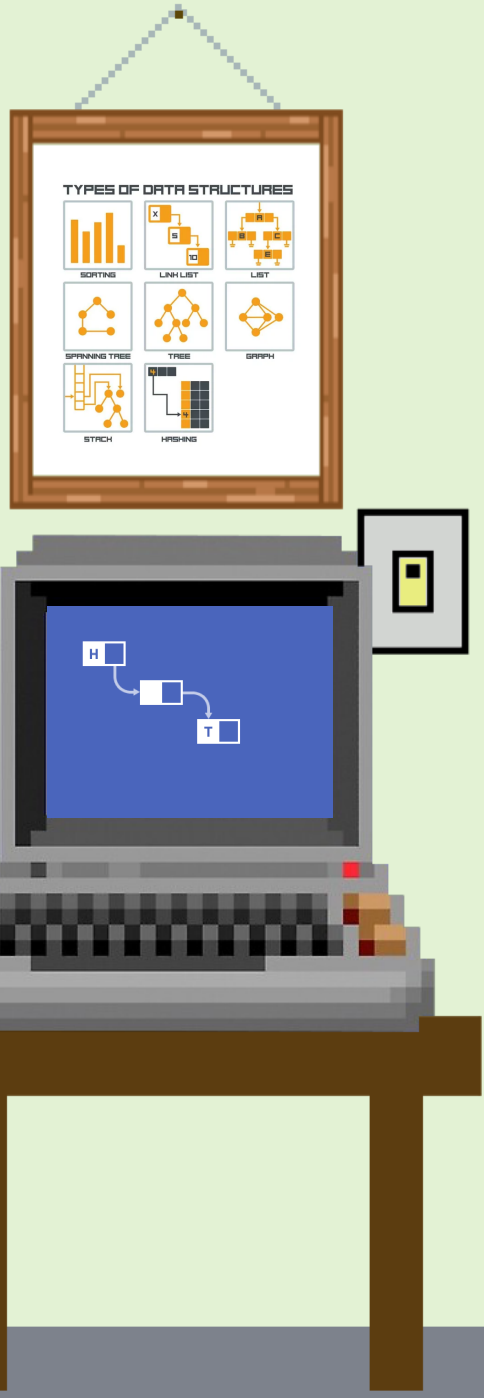






Alocação Dinâmica de Vetores

```
int *v;
int n;
printf("Digite o tamanho do vetor\n");
scanf("%d", &n);
v = malloc (n * sizeof (int));
for (int i = 0; i < n; ++i){
    printf("Digite o valor do %d elemento vetor\n", i);
    scanf ("%d", &v[I]);
}
free (v);
```

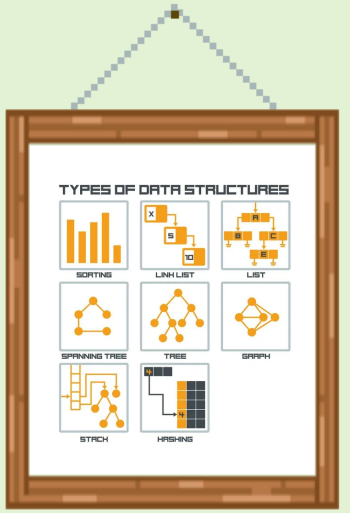


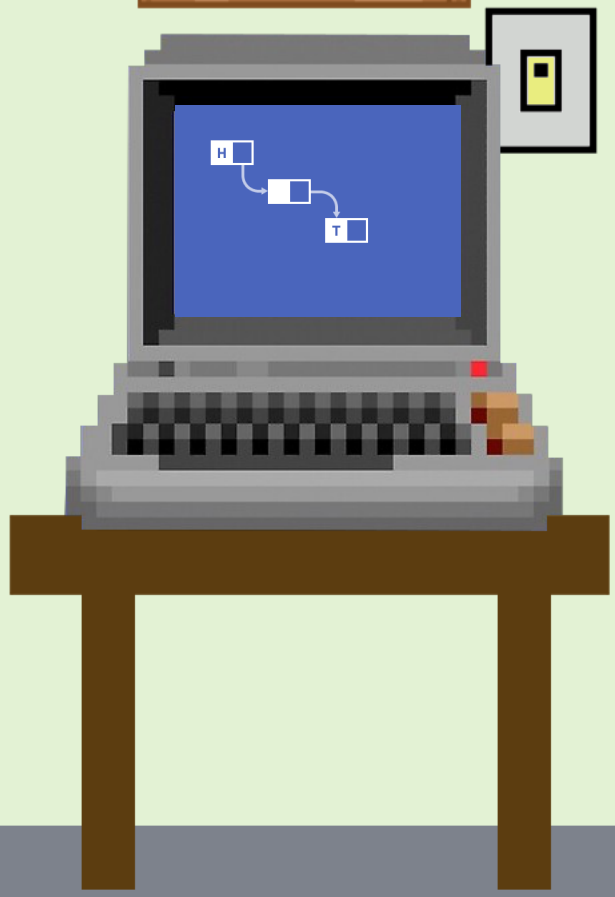
Alocação Dinâmica de Matrizes

```
int **M;
int m,n;

printf("Digite a quantidade de linhas da matriz\n");
scanf("%d", &m);
printf("Digite a quantidade de colunas da matriz\n");
scanf("%d", &n);

M = malloc (m * sizeof (int *));
for (int i = 0; i < m; ++i)
    M[i] = malloc (n * sizeof (int));
```





Atividade

- Resolver a lista de exercícios de revisão

INSTITUTO FEDERAL
Maranhão
Campus Caxias

Curso: Ciência da Computação
Professor: Luís Fernando Maia

Nota

Nome: _____

Disciplina: _____ Data: _____/_____/_____

Lista de Exercícios de Revisão

1. Faça um programa que receba o nome e o sexo de uma pessoa, se for do sexo feminino, o programa irá dizer Bem-vinda + nome da pessoa, se for masculino, Bem-vindo + nome da pessoa.

2. Faça um programa que receba um número entre 1 e 9, e mostre as multiplicações dele como apresentado nas tabuadas.

3. Faça um programa que armazena a nota de 10 alunos e mostre qual a maior e a menor nota.

4. Escreva um programa que declare um inteiro, um float e um char, e ponteiros para inteiro, float, e char. Associe as variáveis aos ponteiros (use &). Modifique os valores de cada variável usando os ponteiros. Imprima os valores das variáveis antes e após a modificação.

5. Elaborar um programa que leia dois valores inteiros (A e B). Em seguida faça uma função que retorne a soma do dobro dos dois números lidos. A função deverá armazenar o dobro de A na própria variável A e o dobro de B na própria variável B.

6. Escreva um programa que declare um array de inteiros e um ponteiro para inteiros. Associe o ponteiro ao array. Agora, some mais um (+1) a cada posição do array usando o ponteiro (use *).

7. Faça um programa que leia um quantidade de elementos n, crie dinamicamente um vetor de n elementos e passe esse vetor para uma função que irá ler seus elementos. Em seguida, o vetor preenchido deve ser impresso.

8. Escreva um programa em linguagem C que solicita do usuário a quantidade de alunos de uma turma e aloca um vetor de notas (números reais). Depois de ler as notas, o programa deve imprimir a média aritmética da turma. Obs: não deve ocorrer desperdício de memória; e após ser utilizada a memória deve ser devolvida.