Estruturas de Dados
(IED-001)

Prof. Dr. Silvio do Lago Pereira

Departamento de Tecnologia da Informação

Faculdade de Tecnologia de São Paulo

Contato

- **Sala**: 623 – Bloco A
- **E-mail**: slagop@gmail.com
- **Página**: www.ime.usp.br/~slago
  - Ementa
  - Bibliografia
  - Critérios de avaliação
  - Cronograma de aulas e provas
  - Compilador Pelles C
  - Notas

Curso

- **Objetivo**: Implementar e usar estruturas de dados na solução de problemas.
- **Tópicos**:
  - Pilhas
  - Filas
  - Recursão
  - Ordenação e busca
  - Listas
  - Mapeamentos
  - Dicionários
  - Árvores

Avaliação

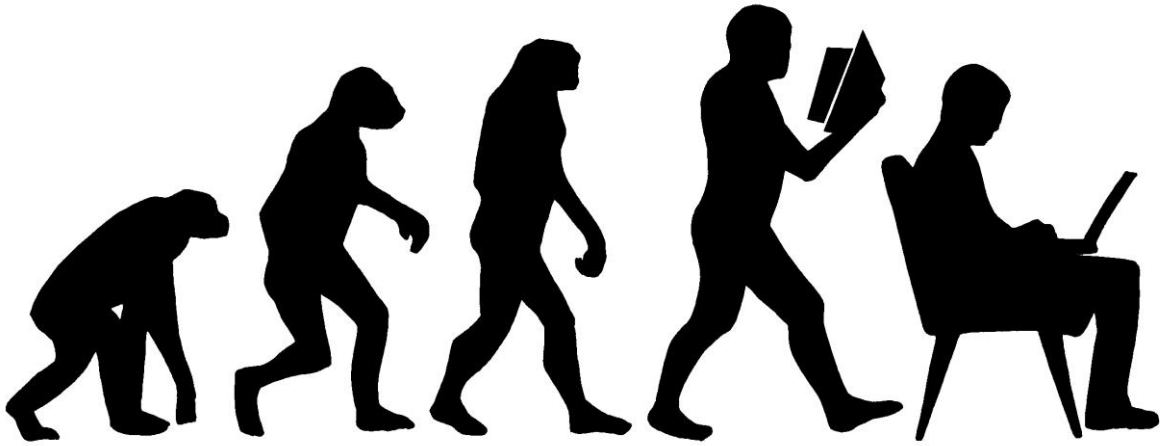
- **Provas**
  - P1
  - P2
  - P3
  - SUB
- **Média** = (P1 + P2 + P3) / 3
  - Aprovação requer média maior ou igual a 6,0.
- **Prova substitutiva**
  - Apenas para quem não atingir a média
  - Substitui a menor nota entre P1, P2 e P3

Introdução
(IED-001)

Introdução

Ciência da Computação

tem como objetivo resolver problemas por meio da criação de **programas** de computador.



- Quando os computadores surgiram, os problemas/programas eram relativamente **simples**.
- Com o aumento da capacidade de armazenamento e processamento dos computadores, os problemas/programas se tornaram **bem mais complexos**.
- Uma forma encontrada para vencer tal complexidade foi aplicar o conceito de **abstração**.

Introdução

Abstração

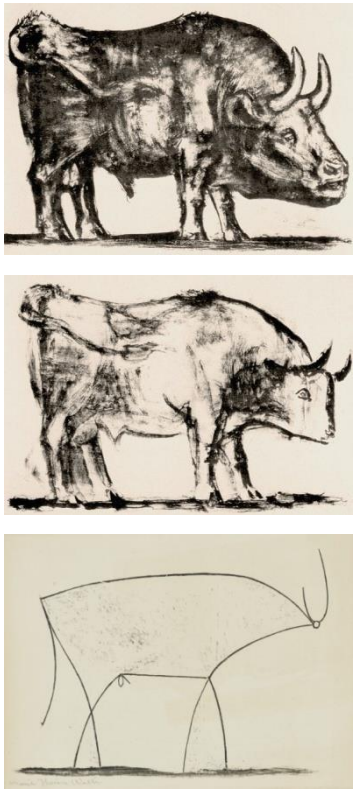
reduz a quantidade de detalhes considerados em cada etapa da criação de um programa.

Abstração funcional:

- Permite que um programador use uma **operação**, sem ter que se preocupar com os detalhes de como ela é executada pelo computador.
- Por exemplo, para usar a função `sqrt()`, só precisamos saber o **que** ela faz; **como** ela faz é irrelevante. Por outro lado, para criarmos essa função, só precisamos saber **como** ela faz; para **que** ela será usada é irrelevante.

Abstração de dados:

- Permite que um programador use um **dado**, sem ter que se preocupar com os detalhes de como ele é armazenado e manipulado pelo computador.
- Por exemplo, para usar um dado do tipo `float`, só precisamos saber **que** operações podem ser feitas com ele; detalhes de **como** ele é armazenado na memória e manipulado pelas operações aritméticas são irrelevantes.



Ao separar o "**que**" do "**como**", a abstração facilita a solução de problemas mais complexos!

Introdução

Uma estrutura de dados

é um tipo de dados abstrato que representa uma coleção de itens inter-relacionados.

Tipo de dados abstrato:

- É composto por um coleção de **dados** e por um conjunto de **operações**.
- Para ser usado, precisa ser **implementado** numa linguagem de programação.

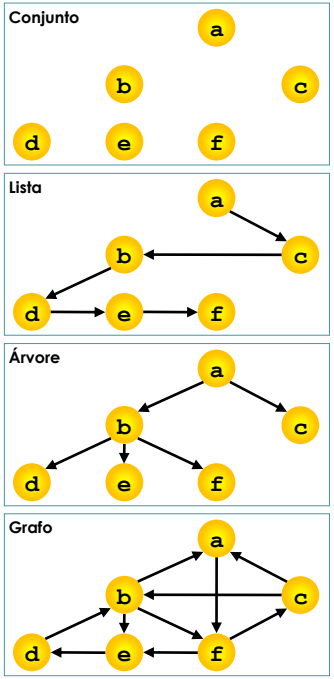
Principais classes de estruturas de dados:

- **Conjunto**: coleção de itens em que não há ordem, nem repetição.
- **Lista**: coleção de itens organizados linearmente.

  Cada item tem um único predecessor e um único sucessor; exceto o *primeiro* item, que não tem predecessor, e o *último*, que não tem sucessor.
- **Árvore**: coleção de itens organizados hierarquicamente.

  Cada item em uma árvore tem um único predecessor e vários sucessores; exceto a *raiz*, que não tem predecessor, e as *folhas*, que não têm sucessores.
- **Grafo:** coleção de itens organizados em rede.

  Cada item em um grafo pode ter vários predecessores e sucessores.



Introdução

A disciplina de Estruturas de dados

estuda formas de organizar dados para serem usados eficientemente pelo computador.

Objetivos da disciplina:

- **Implementar** estruturas de dados específicas em C como, por exemplo:
  - Pilha.
  - Fila.
  - Mapeamento.
  - Dicionário.
- **Usar** estruturas de dados para resolver alguns problemas computacionais específicos como, por exemplo:
  - Manipulação de expressões.
  - Coloração de regiões gráficas.
  - Compactação de dados.

Vamos começar revisando alguns conceitos de programação em linguagem C!

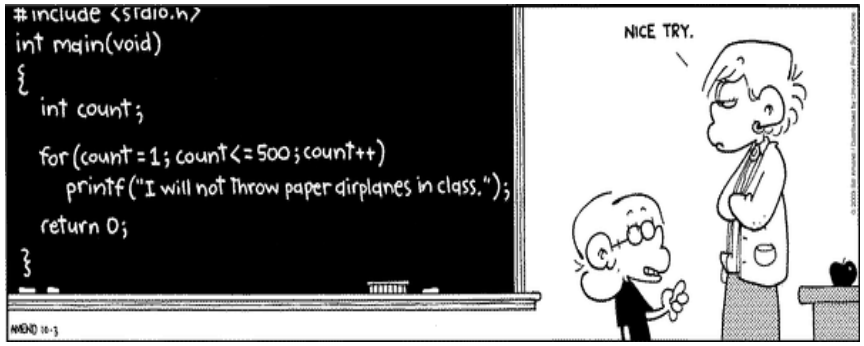
Introdução

Linguagem C

é considerada uma das linguagens de programação mais eficientes que existem.

Função:

- É um bloco de código independente e reusável.
- Um programa C é composto por funções.
- A função principal em C é chamada **`main`**.
- A execução sempre inicia na função principal.



Função principal:

```
int main(void)
{
    ...
    ...
    return 0;
}
```

interface da função

corpo da função

declaração de variáveis

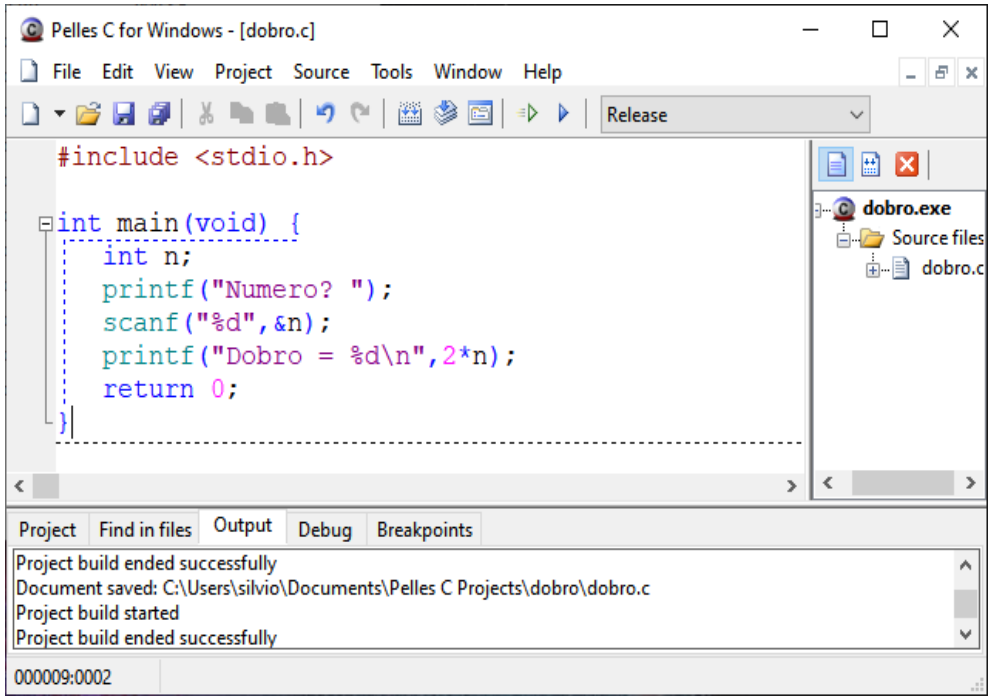
comandos

resposta da função

Introdução

Compilador Pelles C

é o software que usaremos para criar e executar programas codificados em C.



Criação de programa:

- Inicie a execução do **Pelles C IDE**.
- No menu, selecione **File** → **New** → **Project** → **Win32 Console program**.
- Dê um nome ao projeto e clique **OK**.
- Pressione **Ctrl-N** para criar arquivo.
- Digite o código do programa.
- Pressione **Ctrl-S** para salvá-lo (responda ***sim*** à pergunta que é feita).
- Pressione **Ctrl-F5** para compilar/executar o programa.

Esse compilador é de uso livre e está disponível para download na página da disciplina!

Introdução

Tipo de dados

define um conjunto de **valores** e um conjunto de **operações** que podem ser feitas com eles.

Principais tipos em C:

- `char` representa caracteres: `'a'`, `'2'`, `'@'`, `'\n'`, ...
- `int` representa números inteiros: `0`, `2`, `345`, ...
- `float` e `double` representam números reais: `0.1`, `2.5`, `3.45`, ...
- `void` representa ausência de valor.
- Não existe um tipo para ***string***, mas existem constantes desse tipo: `"b"`, `"1"`, `"Ana"`, ...

Principais operadores em C:

- **Atribuição**: `=`.
- **Aritméticos**: `+`, `-`, `/`, `*` e `%`.
- **Relacionais**: `==`, `!=`, `<`, `<=`, `>` e `>=`.
- **Lógicos**: `!`, `&&` e `||`.
- **Endereço**: `&`.

Observações:

- **Divisão**:
  `7/2 == 3` e `7/2.0 == 3.5`.
- **Lógica**:
  Falso é representado por `0` e verdade por `1`.
  Todo valor diferente `0` é considerado verdade.

Todos os operadores em C produzem valores numéricos!

Introdução

Variável

é um **identificador** que representa um **dado** que pode ser modificado pelo programa.

Em C:

- Um **identificador** é uma letra, ou sublinha, seguida de zero ou mais letras, sublinhas e dígitos.
- Antes de usar uma variável devemos **declarar** seu tipo de dados e seu identificador.
- Uma variável pode ser **iniciada**, no momento em que é declarada.

```
char c;
int i = 0;
```

Entrada e saída de dados:

- Declaradas no arquivo `stdio.h` (*standard input/output header*)
- A entrada é feita com a função `scanf("formatação", &variável1, &variável2, ...)`.
- A saída é feita com a função `printf("formatação", valor1, valor2, ...)`.

Caracteres especiais de formatação:

- **Controle**: `\a`, `\b`, `\n`, `\r`, `\t`, `\0`, `\'`, `\"` e `\\`.
- **Formato**: `%c`, `%d`, `%o`, `%x`, `%X`, `%f`, `%lf`, `%%`.

Introdução

Estrutura sequencial

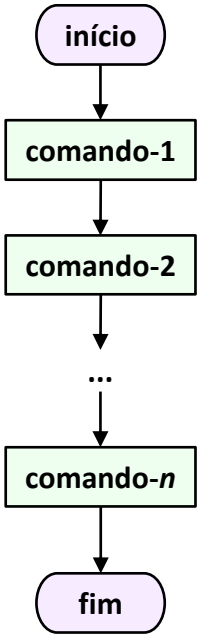
executa uma sequência de comandos, na ordem em que eles são especificados.

Exemplo 1. Índice de massa corporal [1ª versão]

O índice de massa corporal (IMC) de uma pessoa é o seu peso dividido pelo quadrado de sua altura. Crie um programar calcular o IMC de uma pessoa.

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

int main(void) {
   float p, a, i;
   printf("Peso e altura? ");
   scanf("%f %f",&p,&a);
   i = p/pow(a,2);
   printf("IMC = %.2f\n",i);
   return 0;
}
```



A função **pow()** está declarada no arquivo **math.h**!

Introdução

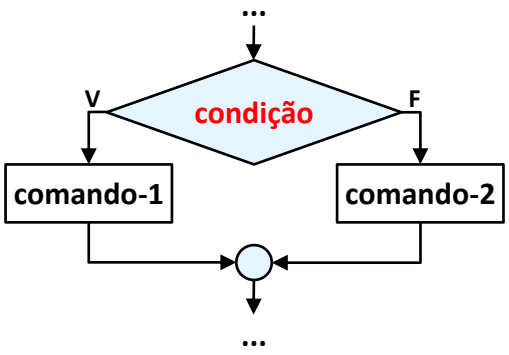
Estrutura de seleção `if-else`

seleciona e executa um “comando”, entre dois comandos alternativos.

Exemplo 2. Índice de massa corporal [2ª versão]

Pessoa com IMC inferior a 18.5 está **magra**, superior a 30 está **obesa** e,caso contrário,está **normal**.Altereo programa com essa informação.

```
#include <stdio.h>
#include <math.h>
int main(void) {
   float p, a, i;
   printf("Peso e altura? ");
   scanf("%f %f",&p,&a);
   i = p/pow(a,2);
   printf("IMC = %.2f\n",i);
   if( i<18.5 ) puts("Magra");
   else if( i>30 ) puts("Obesa");
   else puts("Normal");
   return 0;
}
```




```
if( condição )
    comando-1;
else
    comando-2;
```

O uso de **`else`** com o comando **`if`** é opcional!

Introdução

Estrutura de seleção `switch-case`

Seleciona um caso para continuar a execução, conforme o valor de uma expressão integral.

Exemplo 3. Rodízio de veículos

Dada a placa de um veículo (só dígitos), informe o dia do rodízio.

```
#include <stdio.h>
int main(void) {
   int p;
   printf("Placa? ");
   scanf("%d",&p);
   switch( p%10 ) {
      case 1: case 2: puts("Segunda-feira"); break;
      case 3: case 4: puts("Terca-feira");   break;
      case 5: case 6: puts("Quarta-feira");  break;
      case 7: case 8: puts("Quinta-feira");  break;
      default:        puts("Sexta-feira");
   }
   return 0;
}
```

É uma versão mais restrita de estrutura if-else encadeada!

O uso de **`break`** no final de cada caso é opcional (mas quase sempre necessário)!

Introdução

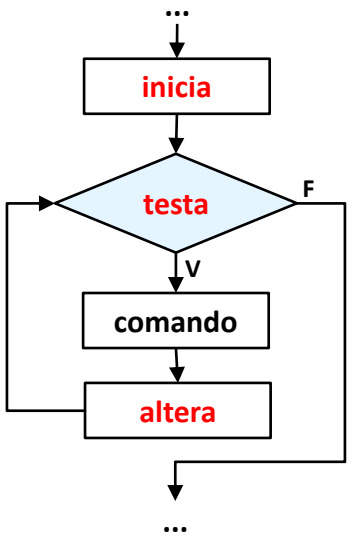
Estrutura de repetição `for`

executa um "comando" um número específico de vezes.

Exemplo 4. Fatorial

Dado um número natural, exiba o seu fatorial.

```
#include <stdio.h>
int main(void) {
    int n, f;
    printf("Numero? ");
    scanf("%d",&n);
    f = 1;
    for(int i=2; i<=n; i++)
        f *= i;
    printf("Fatorial: %d\n",f);
    return 0;
}
```




```
for(inicia; testa; altera)
    comando;
```

Alguns operadores que alteram variáveis acumuladoras em C: `++`, `--`, `+=`, `-=`, `*=`, `/=` e `%=`.

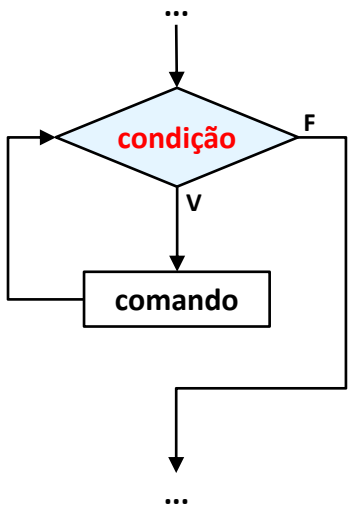
Estrutura de repetição `while`

executa um "comando" zero ou mais vezes.

Exemplo 5. Soma de dígitos

Dado um número natural, exiba a soma de seus dígitos.

```
#include <stdio.h>
int main(void) {
   int n;
   printf("Numero? ");
   scanf("%d",&n);
   int s=0;
   while( n>0 ) {
      s += n%10;
      n /= 10;
   }
   printf("Soma dos digitos = %d\n",s);
   return 0;
}
```




```
while( condição )
    comando ;
```

Usamos bloco (`{ ... }`) para colocar vários comandos onde é esperado um único comando!

Introdução

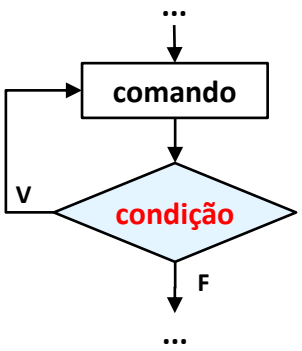
Estrutura de repetição `do-while`

executa um “comando” uma ou mais vezes.

Exemplo 6. Adivinhação

Crie um programa que simule um jogo de adivinhação.

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
int main(void) {
   srand(time(NULL));
   int c, n = rand()%7 + 1;
   do {
      printf("Chute entre 1 e 7: ");
      scanf("%d",&c);
      if( c<n ) puts("Baixo!");
      else if( c>n ) puts("Alto!");
   } while( n!=c );
   puts("Acertou!");
   return 0;
}
```




```
do
    comando ;
while( condição );
```

Introdução

Função

implementa uma operação, ou comando, independente e reusável.

```
tipo  nome (parâmetros)  {
      ...
}
```

- ***nome*** é o identificador da função.
- ***tipo*** é o tipo da saída da função (se não há saída, use `void`).
- ***parâmetros*** são variáveis de entrada da função (ou `void`).

Exemplo 7. A função fatorial

```
#include <stdio.h>
int fat(int n) {
    int f = 1;
    for(int i=2; i<=n; i++) f *= i;
    return f;
}
int main(void) {
    printf("Fatorial do 5: %d\n", fat(5));
    return 0;
}
```

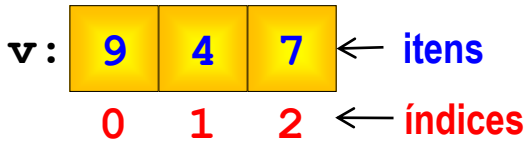
Introdução

Vetor

armazena uma coleção de itens do mesmo tipo.

Criação e iniciação:

```
int v[3] = {9,4,7};
```

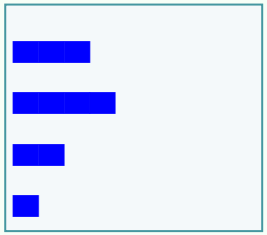



```
v[0] == 9
```

Exemplo 8. Gráfico de barras

```
#include <stdio.h>
void barras(int v[], int n) {
   for(int i=0; i<n; i++) {
      for(int j=0; j<v[i]; j++)
         putchar(220);
      putchar('\n');
   }
}
```

```
int main(void) {
   int a[4] = {3,4,2,1};
   int b[3] = {9,4,7};
   barras(a,4);
   getchar();
   barras(b,3);
   return 0;
}
```



O nome de um vetor representa seu **endereço**; logo, a função tem acesso direto ao vetor original!

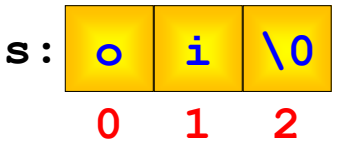
Introdução

String

é um tipo especial de vetor que guarda uma cadeia de caracteres, terminada com `'\0'`.

Criação e iniciação:

```
char s[3] = "oi";
```



`strlen(s)`: dá o comprimento de `s`
`strcpy(a,b)` : copia `b` para `a`
`strcmp(a,b)` : compara `a` com `b`

Exemplo 9. Senha!

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

int main(void) {
    char s[256];
    printf("Senha? ");
    gets(s);
    if( strcmp(s,"abracadabra")==0 ) puts("Ok!");
    else puts("Senha invalida!");
    return 0;
}
```

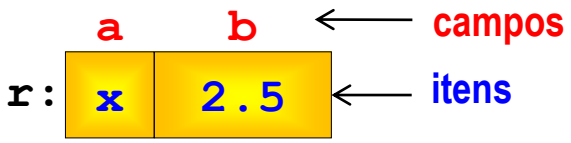
Introdução

Registro

armazena uma coleção de itens, que podem ser de tipos distintos.

Criação e iniciação:

```
typedef struct reg { // cria tipo
   char a;
   float b;
} Reg;
Reg r = {'x',2.5};    // cria variável
```




```
r.a == 'x'
```

Exemplo 10. Ponto!

```
#include <stdio.h>
typedef struct { float x; float y; } Ponto;
int main(void) {
    Ponto p = {1.5,2.5};
    printf("(%.1f,%.1f)\n",p.x,p.y);
    return 0;
}
```

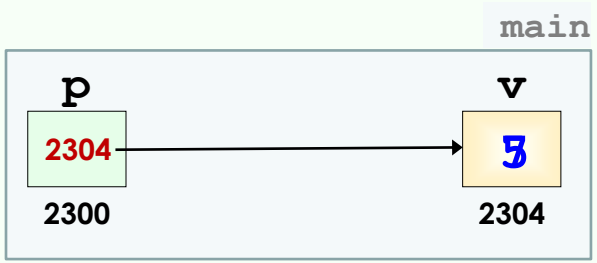
Introdução

Ponteiro

é uma variável que armazena o endereço de outra variável.

Exemplo 11. Funcionamento

```
#include <stdio.h>
int main(void) {
    int v = 5; // variável simples
    int *p;    // variável ponteiro
    p = &v;
    *p = *p + 2;
    printf("v=%d, *p=%d\n",v,*p);
    return 0;
}
```



vídeo

```
v=7, *p=7
```

Passagem de parâmetro :

- **Por valor**: a função recebe o **valor** da variável passada como entrada (usa a **cópia**).
- **Por referência**: a função recebe o **endereço** da variável passada como entrada (usa o **original**).

A passagem de parâmetro por **referência** em C é implementada com **ponteiro**!

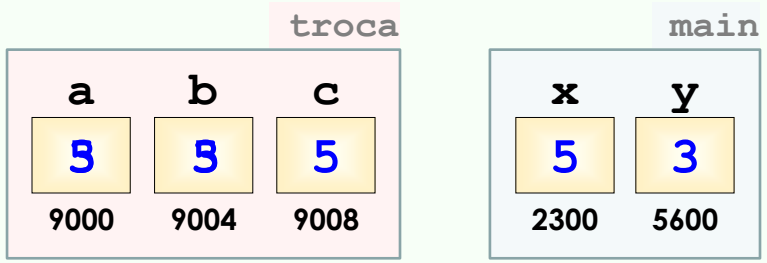
Introdução

Exemplo 12. Passagem por valor

```
void troca(int a, int b) {
    int c = a;
    a = b;
    b = c;
}

troca(x,y);
```

Não produziu o efeito desejado!

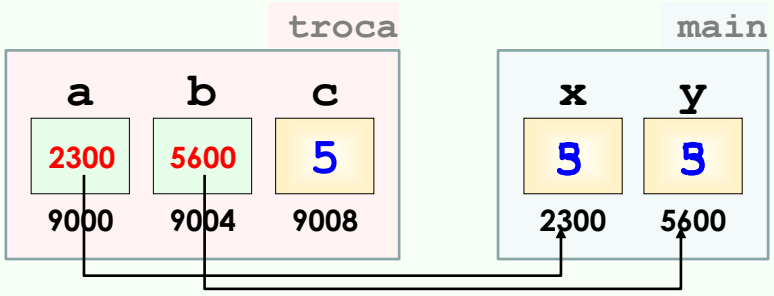


Exemplo 13. Passagem por referência

```
void troca(int *a, int *b) {
    int c = *a;
    *a = *b;
    *b = c;
}

troca(&x,&y);
```

Produziu o efeito desejado!



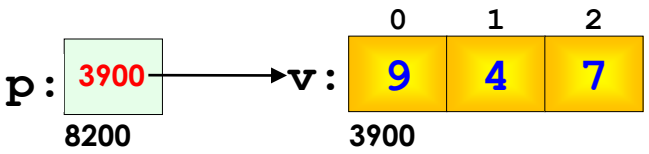
Devemos usar passagem por **referência** sempre que uma função tiver que alterar um parâmetro!

Introdução

É possível criar ponteiro para qualquer tipo de dados:

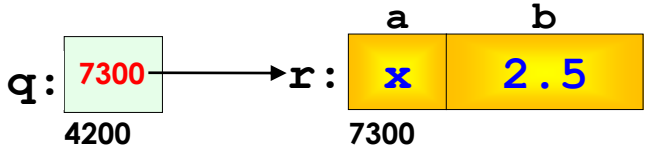
- **Ponteiro para vetor**.

```
int v[3] = {9,4,7};
int *p = v;
printf("%d",p[1]);
printf("%d",*(p+1));
```



- **Ponteiro para registro**.

```
typedef struct {char a; float b; } Reg;
Reg r = {'x',2.5};
Reg *q = &r;
printf("%c",q->a);
printf("%c",(*q).a);
```



Ponteiro nulo:

- Para indicar que um ponteiro não está apontando uma variável, atribua a ele o valor **NULL**.
- A tentativa de acessar o valor apontado por um ponteiro nulo causa um erro de execução fatal.

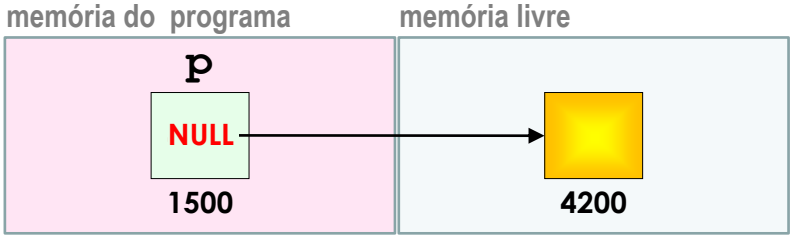
Introdução

Alocação dinâmica

permite criar, em tempo de execução, variáveis que não foram declaradas no programa.

A função `malloc()`:

- Recebe o tamanho da variável a ser criada (em bytes).
- Devolve o endereço da variável criada (ou **`NULL`**, se não houver memória sufiiciente).
- Por exemplo, `int *p = malloc(sizeof(int))` cria uma variável dinâmica do tipo `int`.



A função `free()`:

- Recebe o ponteiro da variável a ser destruída (após a destruição, recomenda-se atribuir **`NULL`**).
- Por exemplo, `free(p)` destrói a variável dinâmica apontada por `p`.

Ambas as funções, bem como a constante **`NULL`**, são declaradas no arquivo **`stdlib.h`**!

Introdução

Alocação dinâmica sequencial

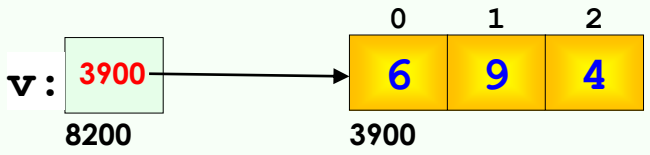
usada para armazenar uma coleção de itens em posições **adjacentes** de memória.

Alocação dinâmica sequencial é feita com **vetores** dinâmicos, criados com `malloc()`.

Exemplo 14. Média de uma sequência de números

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

float media(float v[], int n) {
   float s = 0;
   for(int i=0; i<n; i++)
      s += v[i];
   return s/n;
}
```




```c
int main(void) {
   int n;
   printf("Quantidade de numeros? ");
   scanf("%d",&n);
   float *v = malloc(n*sizeof(float));
   for(int i=0; i<n; i++) {
      printf("%do numero? ",i+1);
      scanf("%f",&v[i]);
   }
   printf("Media = %.2f\n",media(v,n));
   return 0;
}
```

Na alocação dinâmica sequencial o acesso aos itens é rápido, mas inserção e remoção são lentas!

Introdução

Alocação dinâmica encadeada

usada para armazenar uma coleção de itens em posições **arbitrárias** de memória.

Alocação dinâmica encadeada é feita com **registros** dinâmicos, ou **nós**, criados com `malloc()`.

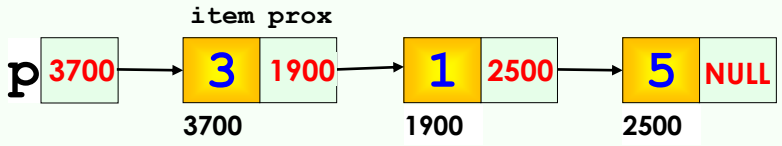
Exemplo 15. Uma lista de números

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

typedef struct no *Ptr;
struct no { int item; Ptr prox; };

Ptr no(int x, Ptr p) {
    Ptr n = malloc(sizeof(struct no));
    n->item = x;
    n->prox = p;
    return n;
}
```

```c
int main(void) {
    Ptr p = no(3,no(1,no(5,NULL)));
    while( p != NULL ) {
        printf("%d\n",p->item);
        p = p->prox;
    }
    return 0;
}
```



Na alocação dinâmica encadeada o acesso aos itens é lento, mas inserção e remoção são rápidas!

Fim