Programação estruturada em

C

Comandos

- Comandos terminam com ponto-e-vírgula
- Comandos de expressão
  - `x = y * z / 2;`
  - `x++;`
  - `printf("texto\n");`
- Comandos de bloco
  - `{ /* lista de comandos */ }`
- Comandos de seleção de fluxo
- Comandos de repetição (*loops*)

Comandos de seleção de fluxo

- Escolhem qual código será executado dependendo do resultado de uma expressão
  - **`SE Ei OP Ej VÁ Ek`**

if (se)

- Executa um comando se uma expressão for verdadeira
- Sintaxe:
  - `if(`***`expressão`***`)` ***`comando`***
- Exemplo:
  - `if(x > 5) x = 5;`
  - ```
    if((x % 2) == 0) {
        if((x % 3) == 0) {
            printf("x eh divisivel por 6\n");
        }
    }
    ```
    ```

if/else

- Executa um comando se uma expressão for verdadeira ou outro comando se for falsa
- Sintaxe:
  - `if(`***`expressão`***`)` ***`comando1`***
  - `else` ***`comando2`***
- Exemplo:
  - ```
    if((x % 2) == 0) { printf("par\n"); }
    else { printf("impar\n"); }
    ```
    ```

if/else

```
if(n > 0)
    if((n % 2) == 0)
        printf("n eh positivo e par\n");
    else
        printf("n eh positivo e impar\n");
```

```
if(n > 0) {
    if((n % 2)
        printf("n eh
    }
    else
        pri
```

```
if(n > 0) {
    if((n % 2) == 0)
        printf("n eh positivo e par\n");
}
else
    printf("n eh negativo ou zero\n");
```

switch

- Escolhe um fluxo entre vários possíveis dado o valor de uma expressão
- Sintaxe:
  - `switch(`***`expressão inteira`***`)` ***`comando`***
- Para cada valor possível da *expressão*, o bloco do switch deve definir uma sequência de comandos

switch

```
switch(menu()) {
    case 1:
        abrir_caixa();
        break;
    case 2:
        processar_venda();
        break;
    case 3:
        fechar_caixa();
        break;
    case 4:
        atualizar_estoque();
        break;
    default:
        mostrar_ajuda();
}
```

Sintaxe:

```
case constante: comando
default: comando
```

switch

```
switch(operacao) {
    case '+':
        resultado = x + y;
        break;
    case '-':
        resultado = x - y;
        break;
    case '*':
        resultado = x * y;
        break;
    case '/':
        resultado = x / y;
        break;
    default:
        mostrar_ajuda();
        resultado = 0;
}
```

switch

```
switch(menu()) {
    case 'A':
    case 'a':
        operacao_a();
        break;
    case 'B':
    case 'b':
        operacao_b();
        break;
    ...
    ...
    default:
        mostrar_ajuda();
}
```

Exemplo

- `calculadora`

Comandos de repetição

- Facilitam a criação de laços para repetir a execução de comandos várias vezes

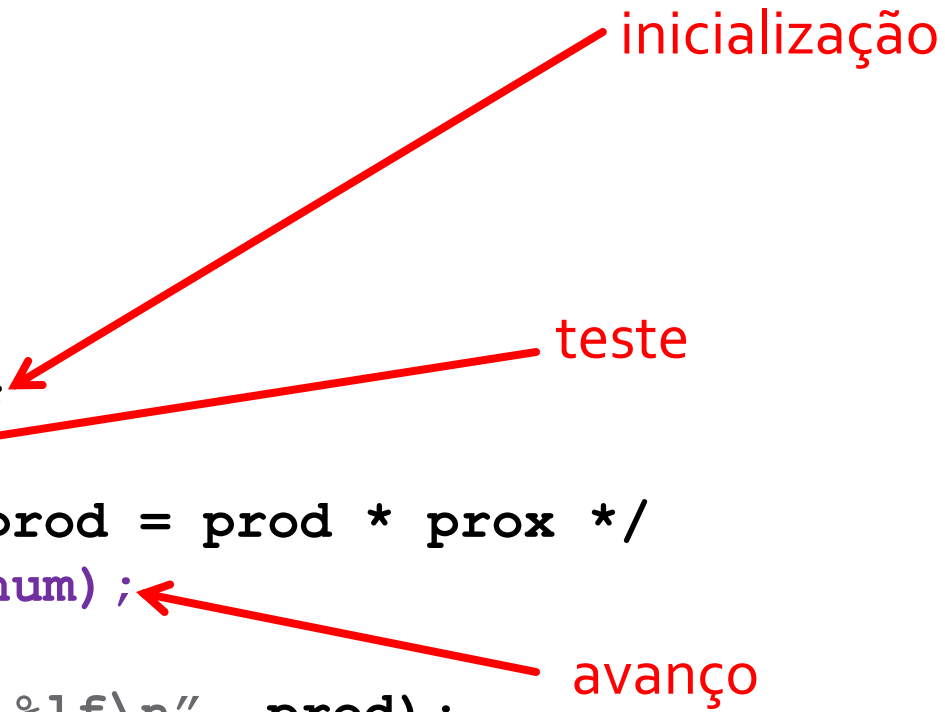
- Exemplos:
  - Cálculo de produtório: ler vários números até ler um zero
  - Vários problemas precisam executar uma operação sobre várias entradas

while

- Repete um comando enquanto uma expressão for verdadeira
- Sintaxe:
  - `while(`***`expressão`***`)` ***`comando`***

while

```
#include <stdio.h>

int main(void) {
    double prod = 1;
    double num;
    scanf("%lf\n", &num);
    while(num != 0){
        prod *= num; /* prod = prod * prox */
        scanf("%lf\n", &num);
    }
    printf("produtorio = %lf\n", prod);
}
```



while

- Às vezes, um laço infinito é desejável

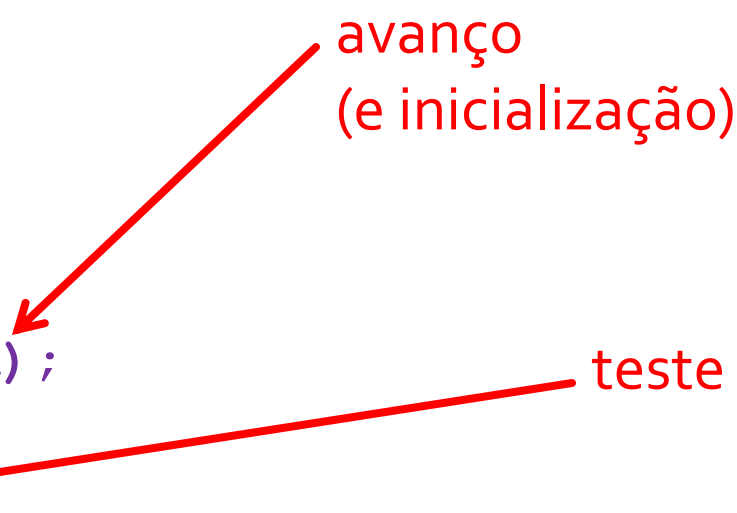
```
while(1) {
    le_comando();
    processa_comando();
}
```

do/while

- Executa um comando enquanto uma expressão é verdadeira
  - Igual ao while, mas executa o comando pelo menos uma vez antes de testar a expressão
  - Menos comum que while
- Sintaxe:
  - `do` *`comando`* `while(`*`expresão`*`);`

do/while

```
#include <stdio.h>

int main(void) {
    double soma = 0;
    double num;
    do {
        scanf("%lf\n", &num);
        soma += num;
    } while(num != 0);
    printf("somatorio = %lf\n", prod);
}
```



for

- Repete um comando enquanto uma expressão é verdadeira, mas carrega junto expressões de inicialização e avanço
- Sintaxe:
  - `for(inicio; teste; avanço) comando`
  - *inicío* > *teste* > *comando* > *avanço* > *teste* > *comando* > *avanço* ...
  - Coloca todas as informações do controle de repetições em um local
  - Expressões podem ser vazias

for

```
#include <stdio.h>

int main(void) {
    double prod = 1;
    double num;
    scanf("%lf\n", &num);
    while(num != 0) {
        prod *= num;
        scanf("%lf\n", &num);
    }
    printf("prod = %lf\n", prod);
}
```

for

```
#include <stdio.h>

int main(void) {
    double prod = 1;
    double num;
    for(scanf("%lf\n", &num); num != 0;
            scanf("%lf\n", &num)) {
        prod *= num;
    }
    printf("prod = %lf\n", prod);
}
```

Controle de repetições

- Existem dois comandos que podemos executar para controlar repetições dentro de while, do/while e for

- break
  - sai do comando de repetição
- continue
  - cancela a iteração atual e inicía a próxima

break

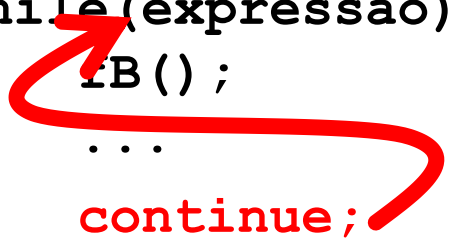
```
#include <stdio.h>

int main(void) {
    ...
    fA();
    while(expressao) {
        fB();
        ...
        break;
        ...
        fX();
    }
    fZ();
    ...
}
```

break

```
#include <stdio.h>

int main(void) {
    ...
    fA();
    for(inicio(); teste(); avanco()) {
        fB();
        ...
        break;
        ...
        fX();
    }
    fZ();
    ...
}
```

continue

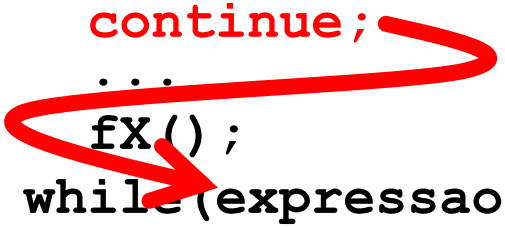
```
#include <stdio.h>

int main(void) {
    ...
    fA();
    while(expressao) {
        fB();
        ...
        continue;
        ...
        fX();
    }
    fZ();
    ...
}
```



Depois do continue, testamos a expressão antes de executar a próxima iteração.

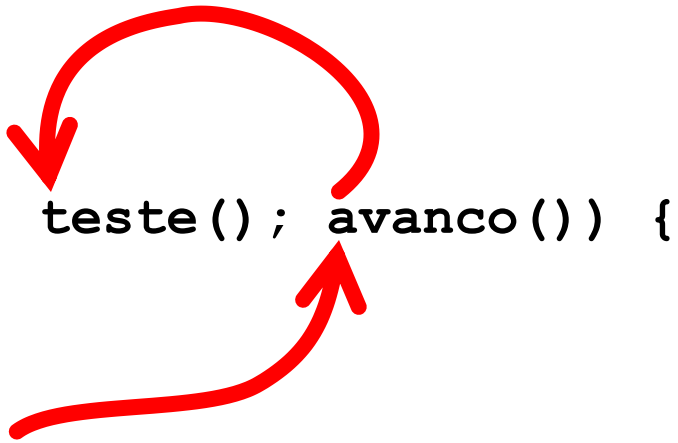
continue

```
#include <stdio.h>

int main(void) {
    ...
    fA();
    do {
        fB();
        ...
        continue;
        ...
        fX();
    } while(expressao);
    fZ();
    ...
}
```



continue

```
#include <stdio.h>

int main(void) {
    ...
    fA();
    for(inicio(); teste(); avanco()) {
        fB();
        ...
        continue;
        ...
        fX();
    }
    fZ();
    ...
}
```



Laços aninhados

- Podemos ter um laço dentro de outro
- Os comandos break e continue aplicam-se ao while "mais próximo"