Programação

C++

Aula 02

Jorgiano Vidal

Agenda

- Funções
- Condicional
  - `if`
  - `if…else`
- Laços
  - `while`
  - `do..while`
  - `for`
  - `continue`
  - `break`

Funções

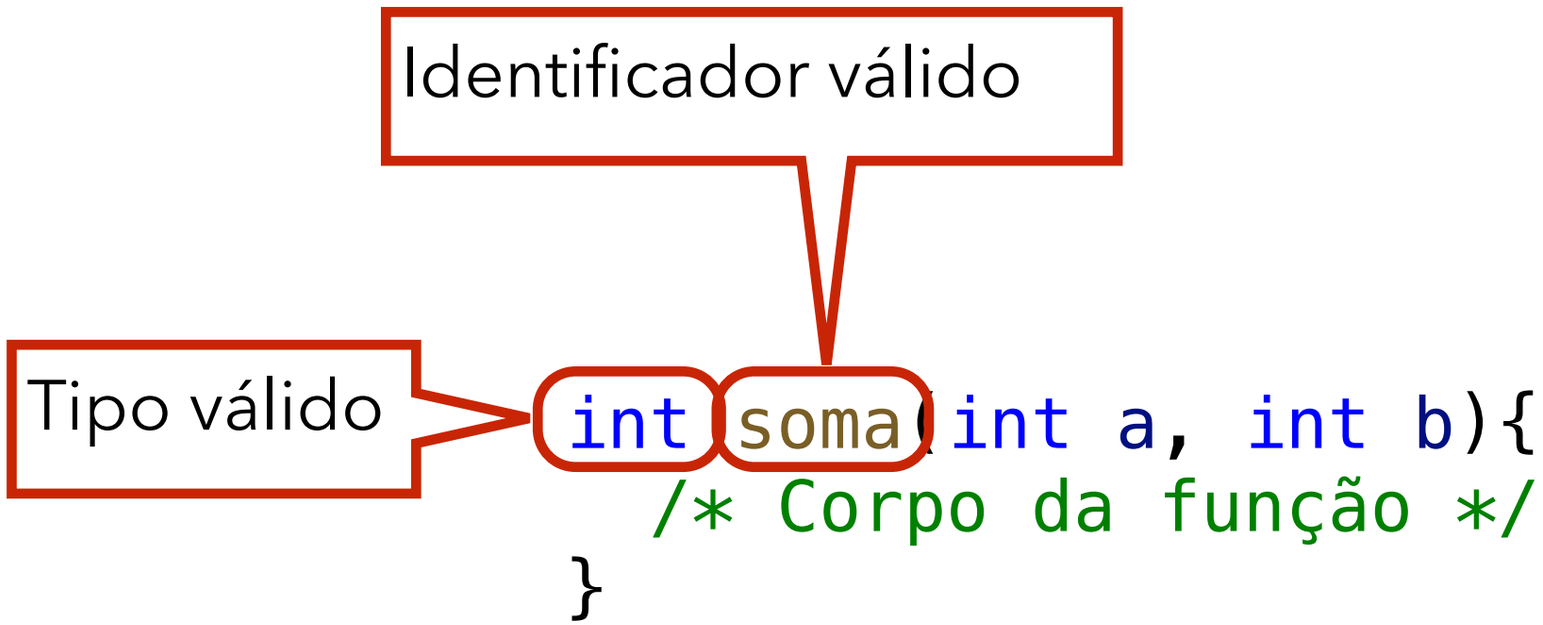
```
int soma(int a, int b){
  /* Corpo da função */
}
```

Funções

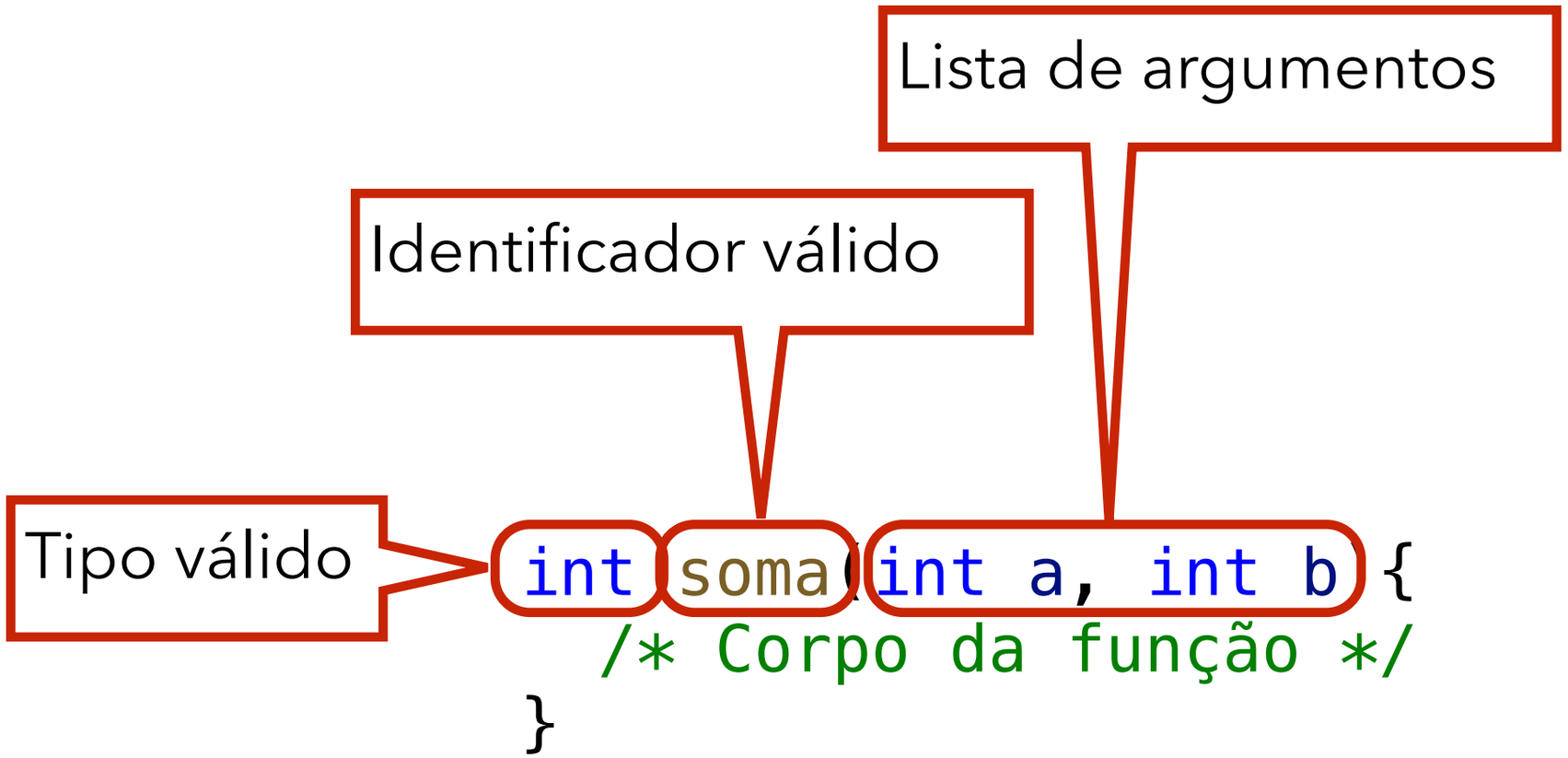
Tipo válido

```
int soma(int a, int b){
   /* Corpo da função */
}
```

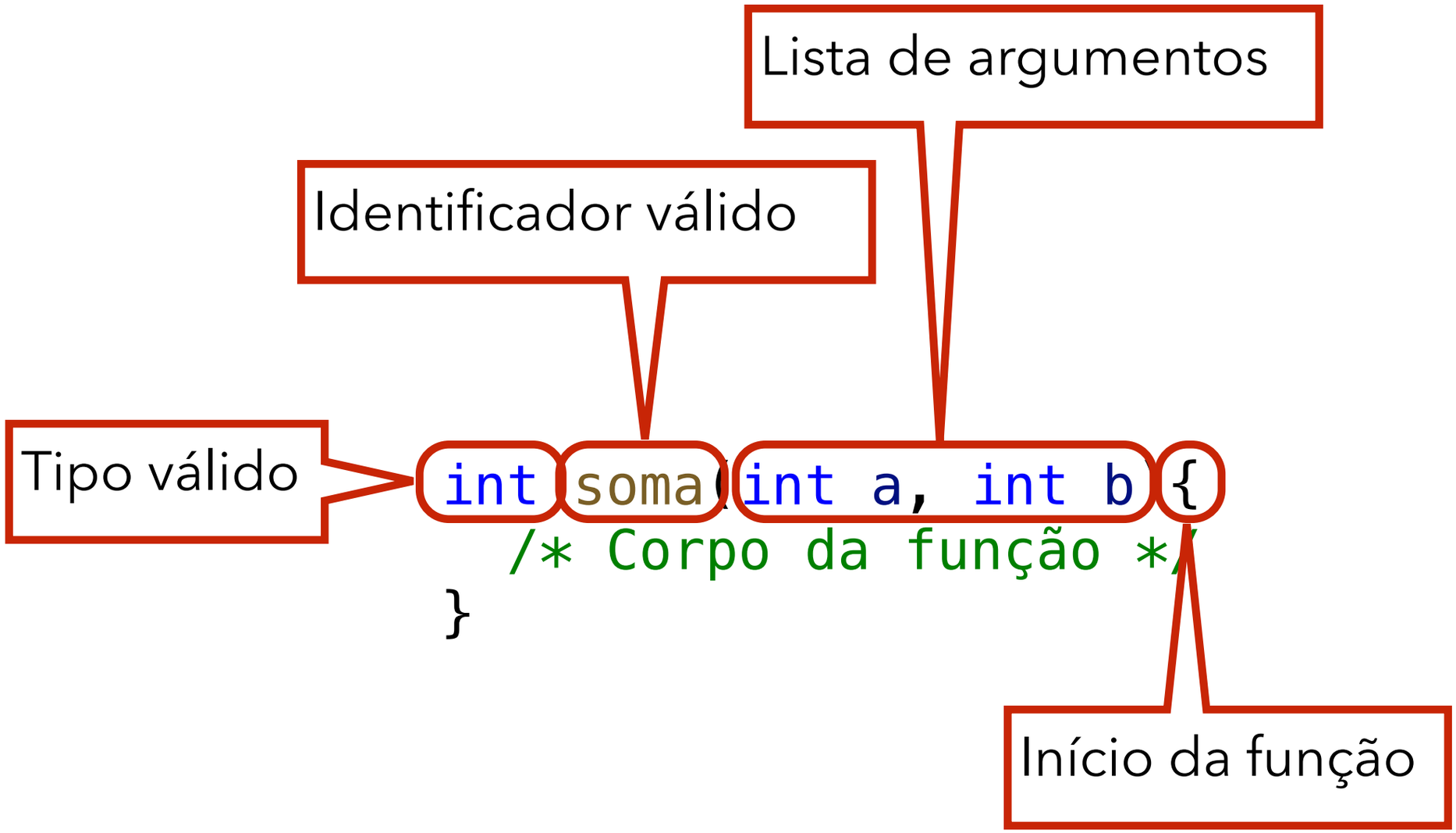
Funções


Identificador válido
Tipo válido
int soma(int a, int b){
/* Corpo da função */
}

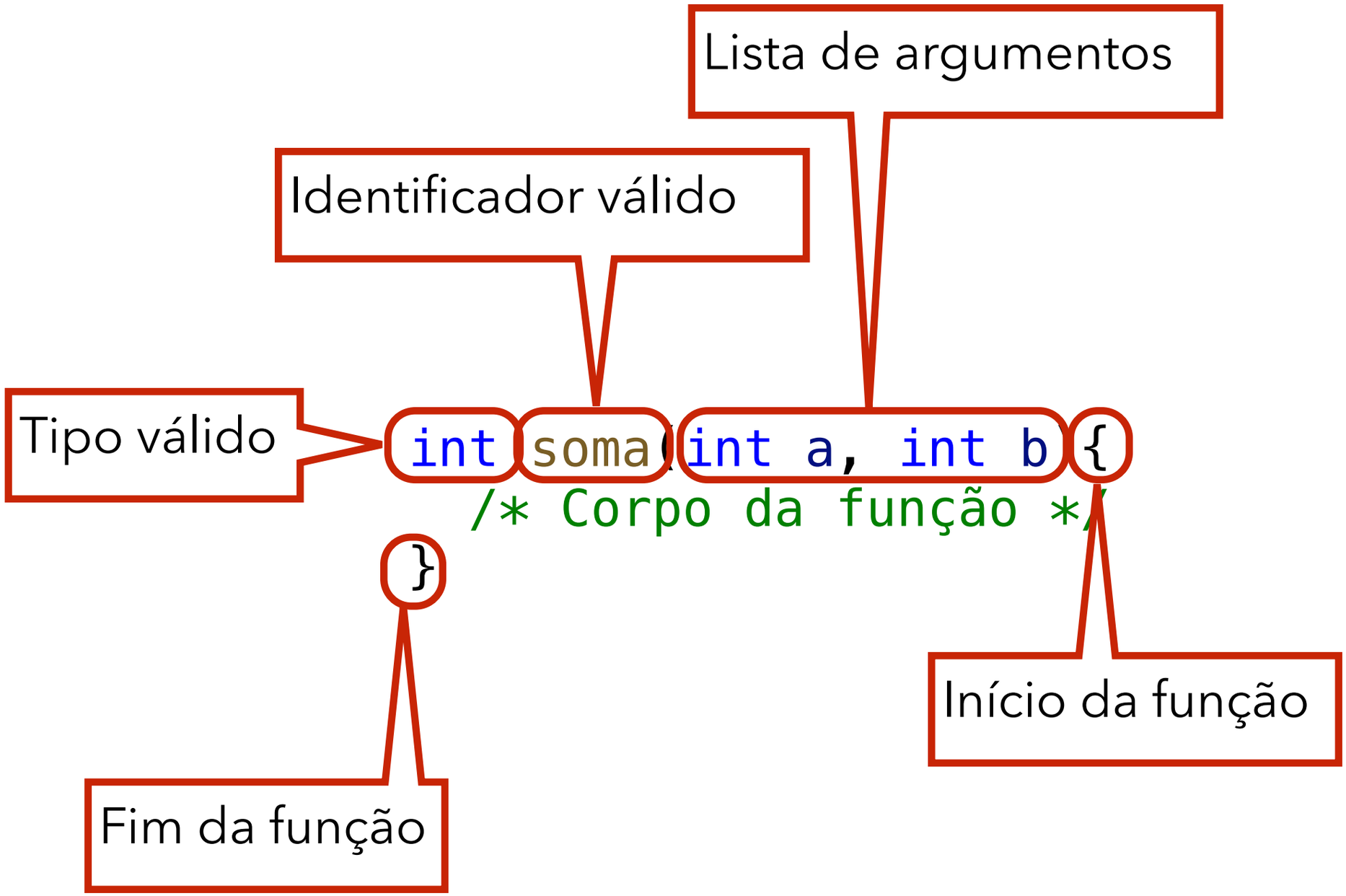
Funções


Lista de argumentos
Identificador válido
Tipo válido
int soma(int a, int b) {
/* Corpo da função */
}

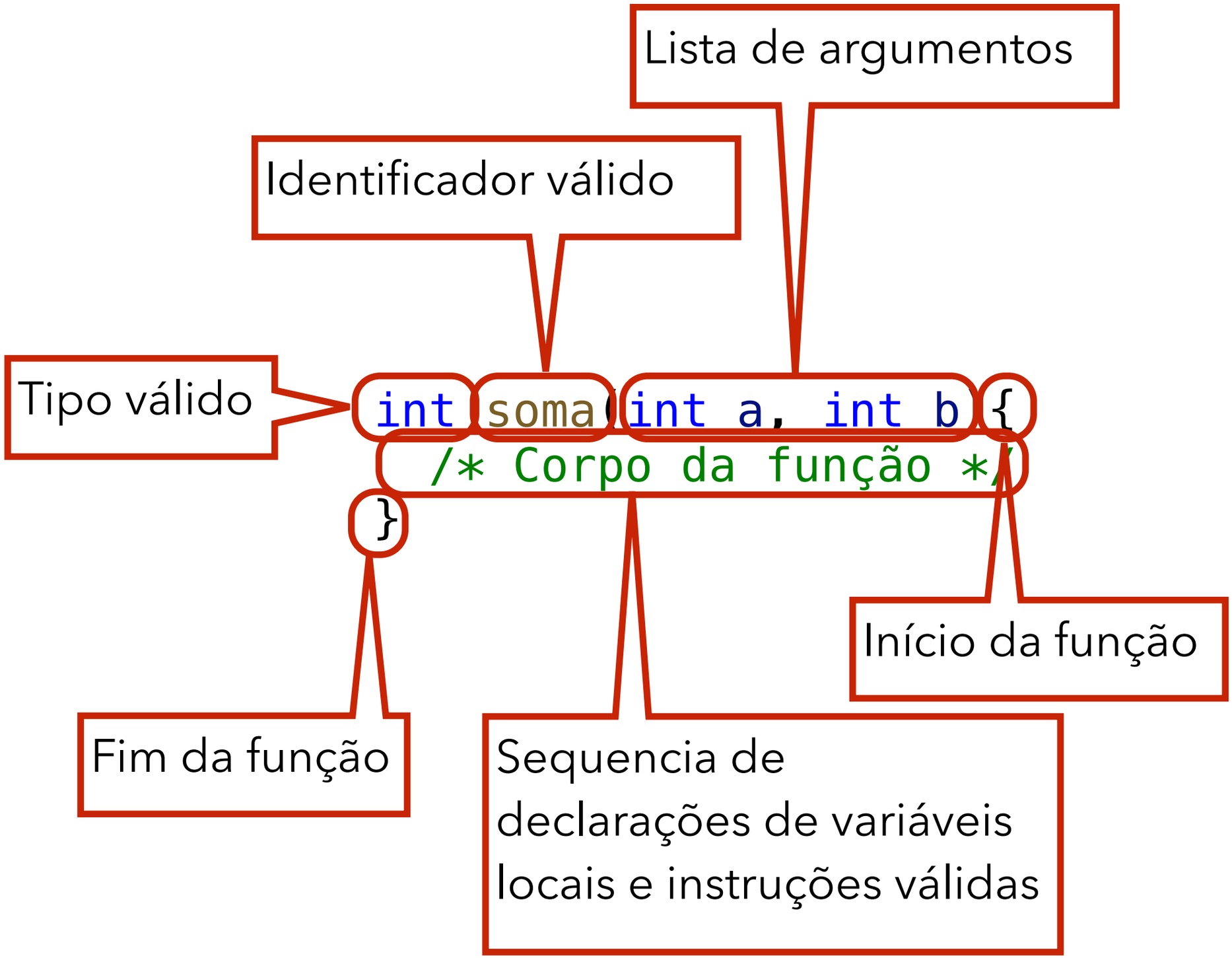
Funções


Lista de argumentos
Identificador válido
Tipo válido
int soma(int a, int b {
/* Corpo da função */
}
Início da função

Funções


Lista de argumentos
Identificador válido
Tipo válido
int soma(int a, int b {
/* Corpo da função */
}
Início da função
Fim da função

Funções


Lista de argumentos
Identificador válido
Tipo válido
int soma(int a, int b) {
/* Corpo da função */
}
Início da função
Fim da função
Sequencia de declarações de variáveis locais e instruções válidas

Corpo da função

- Declarações e instruções
- Ponto-e-vírgula é separador
- Formatação (e indentação) é essencial para compreensão do código, mas indiferente para o compilador

```
int soma(int a, int b){
   int s;
   s = a+b;
   return s;
}
```

Corpo da função

- Declarações e instruções
- Ponto-e-vírgula é separador
- Formatação (e indentação) é essencial para compreensão do código, mas indiferente para o compilador

```
int soma(int a, int b){
  int s;
  s = a+b;
  return s;
}
```

Declarações

Instruções

Corpo da função

- Declarações e instruções
- Ponto-e-vírgula é separador
- Formatação (e indentação) é essencial para compreensão do código, mas indiferente para o compilador

```
int soma(int a, int b){
  int s;
  s = a+b;
  return s;
}
```

Declarações

Instruções

```
int
soma(
int a,
int b)
{
int s; s=a+b;
return
s;
}
```

```
int soma(int a, int b) {int s; s=a+b;return s;}
```

Exemplo

```cpp
#include <iostream>

int soma(int a, int b) {
   int r;
   r = a + b;
   return r;
}

int main() {
   int a, b, s;
   std::cin >> a >> b;
   s = soma(a, b);
   std::cout << "A soma é "
             << s
             << std::endl;
   return 0;
}
```

Condicional

- Comando if

```
if (expr) stmt; [else stmt;]
```

- stmt pode ser composto

```
if (expr) {stmt_seq;} [else {stmt_seq;}]
```

Condicional

- Comando if

```
if (expr) stmt; [else stmt;]
```

- stmt pode ser composto

```
if (expr) {stmt_seq;} [else {stmt_seq;}]
```

```
#include <iostream>

int main (){
  double nota1, nota2, media;
  std::cin >> nota1 >> nota2;
  media = (nota1+nota2)/2;
  if ( media >= 7.0 )
    std::cout << "Aluno aprovado!\n";
  else
    std::cout << "Aluno em recuperação!\n";

  return 0;
}
```

Condicional

- Comando if

```
if (expr) stmt; [else stmt;]
```

- stmt pode ser composto

```
if (expr) {stmt_seq;} [else {stmt_seq;}]
```

```cpp
#include <iostream>

int main (){
  double nota1, nota2, media;
  std::cin >> nota1 >> nota2;
  media = (nota1+nota2)/2;
  if ( media >= 7.0 ){
    std::cout << "Aluno aprovado!\n";
  } else {
    std::cout << "Aluno em recuperação!\n";
  }
  return 0;
}
```

Operadores relacionais

- Compara valores

Operador	Sintaxe	Exemplo
Igual	`==`	`if (x==y)`
Diferente	`!=`	`if (a!=(b-c))`
Maior	`>`	`if ((a+b)>(c+d))`
Menor	`<`	`if ((x*y)<d)`
Maior igual	`>=`	`if (z>=(y+10))`
Menor igual	`<=`	`if (5<=(x-3))`

Operadores lógicos

Operador	Sintaxe	Exemplo
E lógico	`&&` ou `and`	`if (a>3 && b<c)` `if (a>3 and b<c)`
OU lógico	`||` ou `or`	`if (media<7 || faltas>20)` `if (media<7 or faltas>20)`
Negação lógica	`!` ou `not`	`if (!(a==b) )` `if (not(a==b))`

Laços

- Três instruções
  - `while`
  - `do…while`
  - `for`
- O **`goto`** também existe, porém seu uso, de forma geral, é desaconselhável
- Usados em laços
  - `break`
  - `continue`

while

while

- Repete instruções enquanto teste (expressão) for verdadeiro
- Sintaxe:

```
while (expr) stmt;
```

while

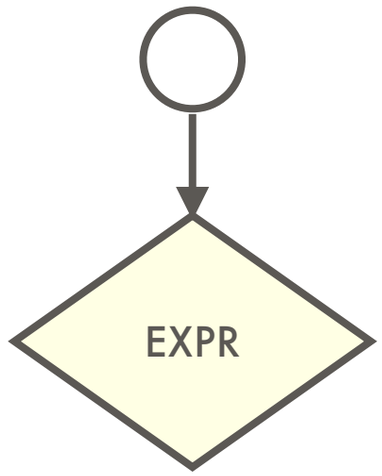
- Repete instruções enquanto teste (expressão) for verdadeiro
- Sintaxe:

```
while (expr) stmt;
```

while

- Repete instruções enquanto teste (expressão) for verdadeiro
- Sintaxe:

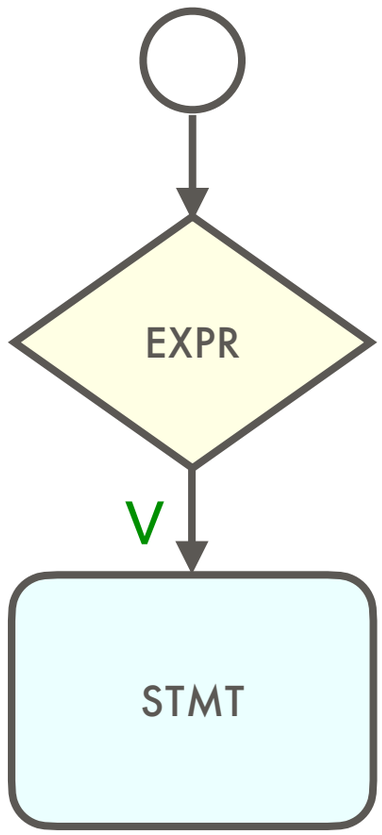
```
while (expr) stmt;
```



while

- Repete instruções enquanto teste (expressão) for verdadeiro
- Sintaxe:

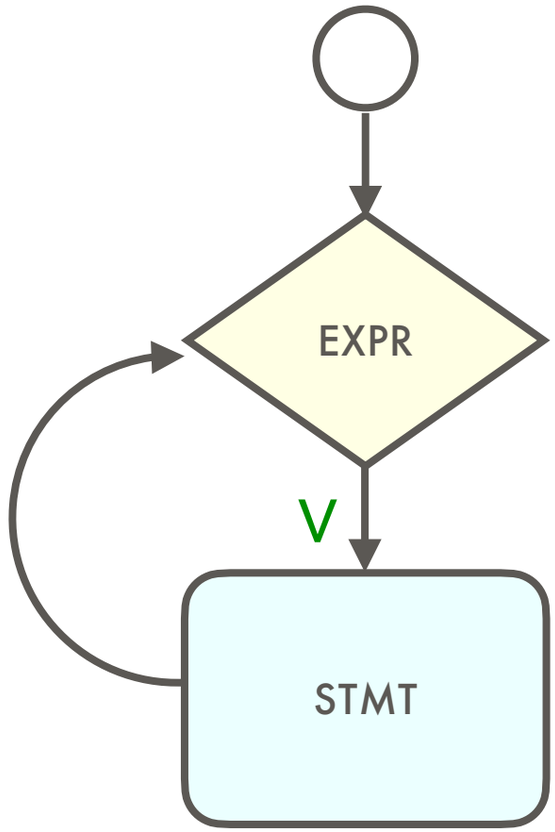
```
while (expr) stmt;
```



while

- Repete instruções enquanto teste (expressão) for verdadeiro
- Sintaxe:

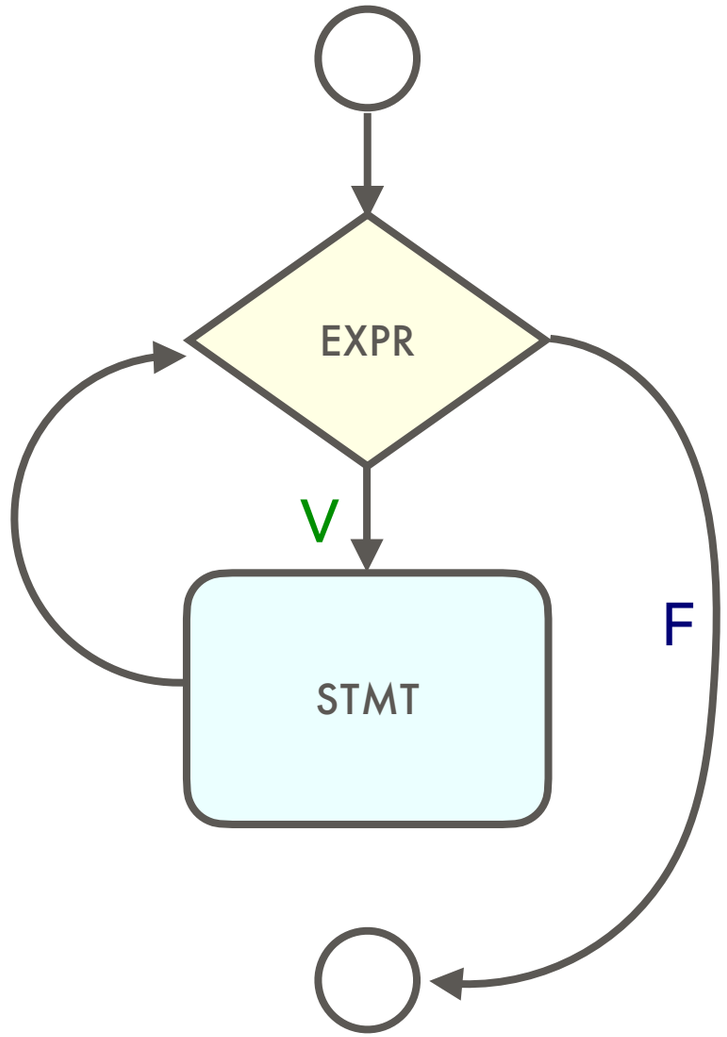
```
while (expr) stmt;
```



while

- Repete instruções enquanto teste (expressão) for verdadeiro
- Sintaxe:

```
while (expr) stmt;
```

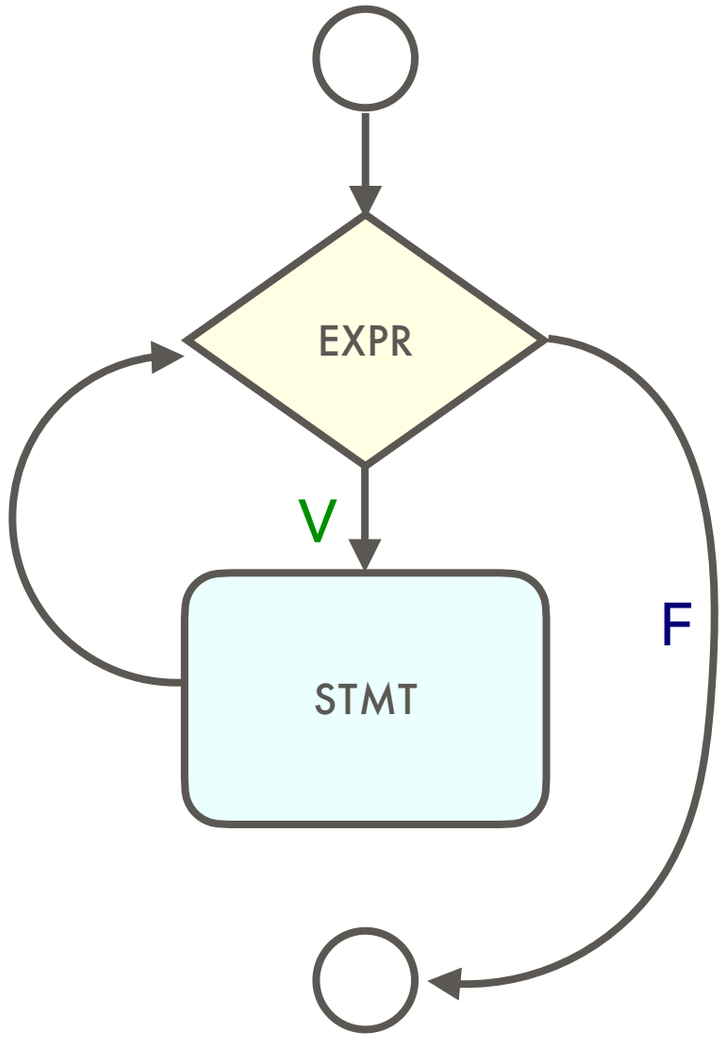


while

- Repete instruções enquanto teste (expressão) for verdadeiro
- Sintaxe:

```
while (expr) stmt;
```

- Exemplo: Contar divisores de n



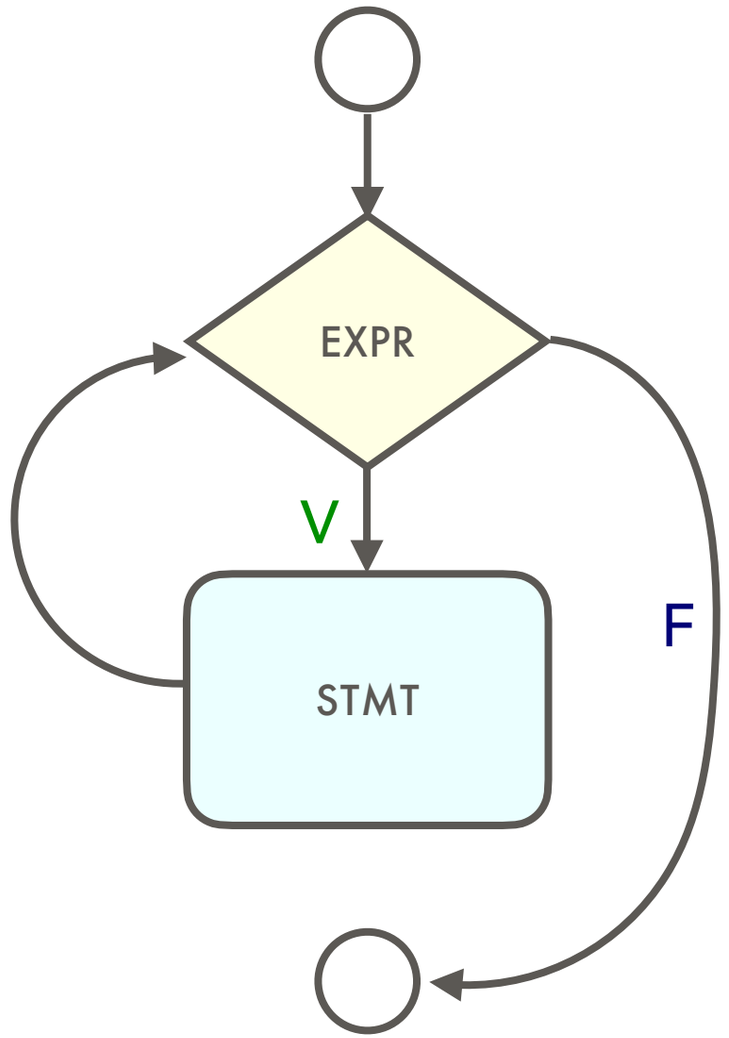
while

- Repete instruções enquanto teste (expressão) for verdadeiro
- Sintaxe:

```
while (expr) stmt;
```

- Exemplo: Contar divisores de n

```
int contar_divisores(int n){
   int d=1,qtd=1;
   while (d!=n){
      if ((n%d)==0)
         qtd=qtd+1;
      d=d+1;
   }
   return qtd;
}
```



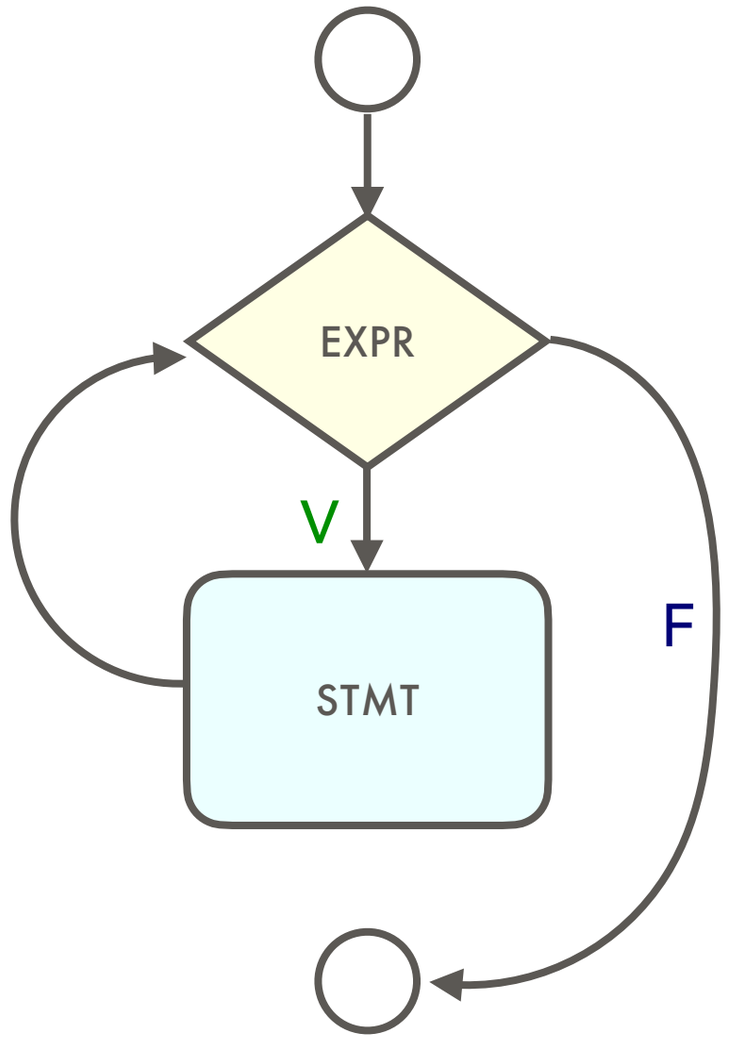
while

- Repete instruções enquanto teste (expressão) for verdadeiro
- Sintaxe:

```
while (expr) stmt;
```

- Exemplo: Contar divisores de n

```
int contar_divisores(int n){
   int d=1,qtd=1;
   while (d!=n){
      if ((n%d)==0)
         qtd=qtd+1;
      d=d+1;
   }
   return qtd;
}
```



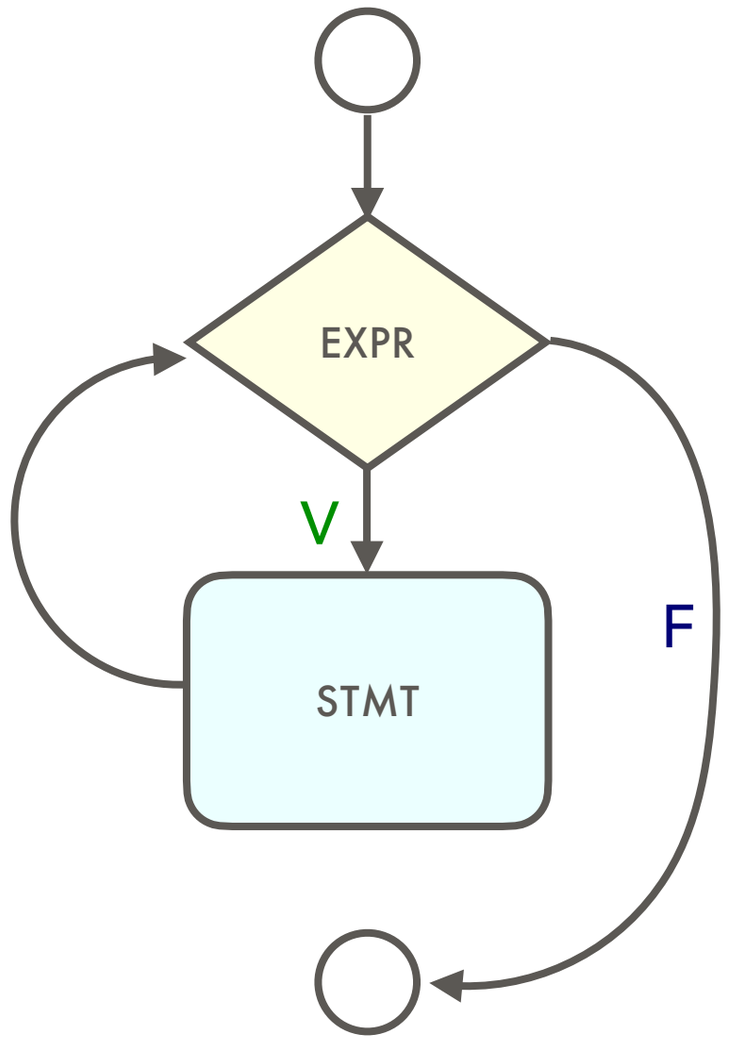
while

- Repete instruções enquanto teste (expressão) for verdadeiro
- Sintaxe:

```
while (expr) stmt;
```

- Exemplo: Contar divisores de n

```
int contar_divisores(int n){
   int d=1,qtd=1;
   while (d!=n){
      if ((n%d)==0)
         qtd=qtd+1;
      d=d+1;
   }
   return qtd;
}
```



IMPORTANTE: Corpo deve modificar resultado da expressão

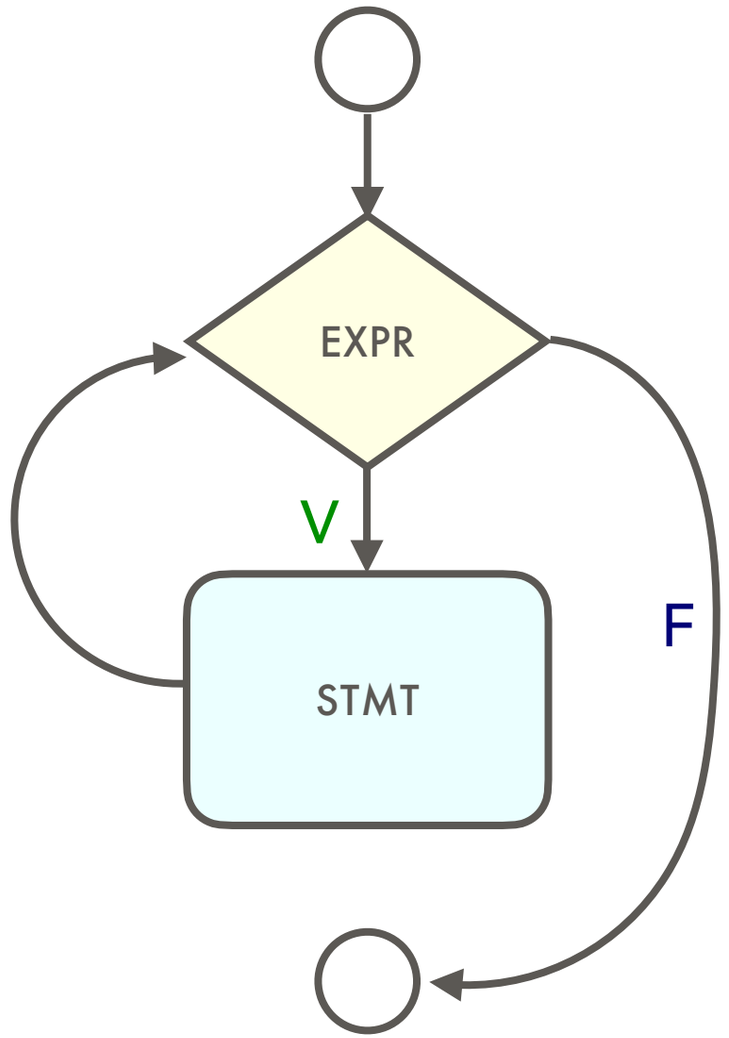
while

- Repete instruções enquanto teste (expressão) for verdadeiro
- Sintaxe:

```
while (expr) stmt;
```

- Exemplo: Contar divisores de n

```
int contar_divisores(int n){
  int d=1,qtd=1;
  while (d!=n){
    if ((n%d)==0)
      qtd=qtd+1;
    d=d+1;
  }
  return qtd;
}
```



IMPORTANTE: Corpo deve modificar resultado da expressão

while

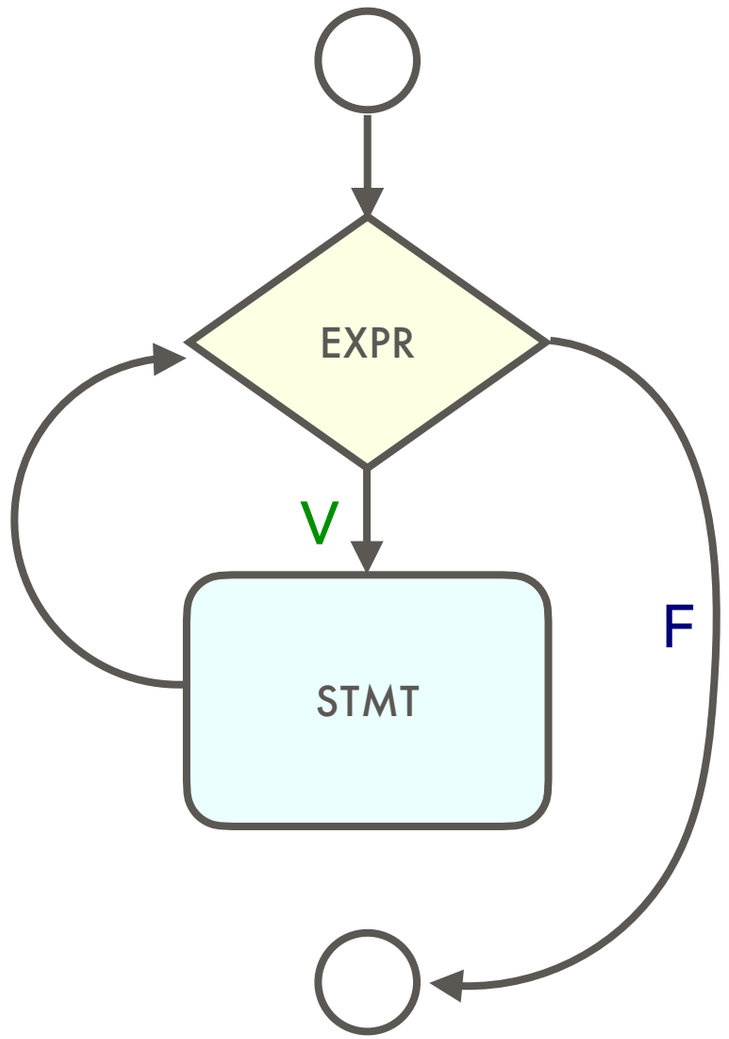
- Repete instruções enquanto teste (expressão) for verdadeiro
- Sintaxe:

```
while (expr) stmt;
```

- Exemplo: Contar divisores de n

```
int contar_divisores(int n){
  int d=1,qtd=1;
  while (d!=n){
    if ((n%d)==0)
      qtd=qtd+1;
    d=d+1;
  }
  return qtd;
}
```

qtd=qtd+1; → qtd++;



IMPORTANTE: Corpo deve modificar resultado da expressão

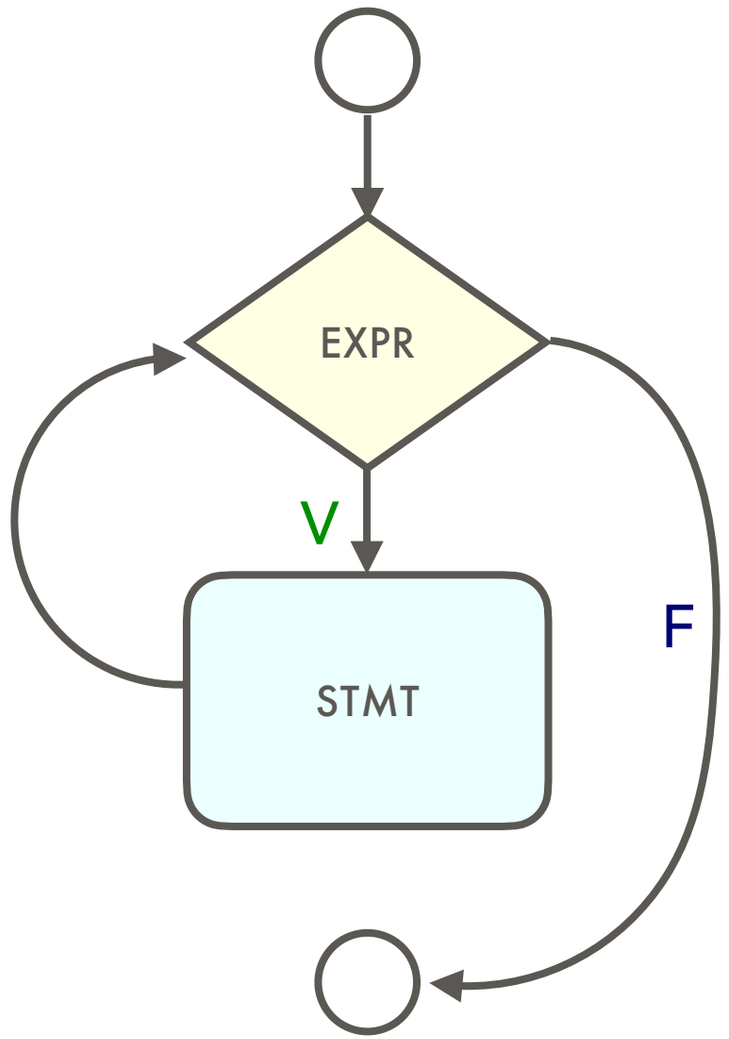
while

- Repete instruções enquanto teste (expressão) for verdadeiro
- Sintaxe:

```
while (expr) stmt;
```

- Exemplo: Contar divisores de n

```
int contar_divisores(int n){
  int d=1,qtd=1;
  while (d!=n){
    if ((n%d)==0)
      qtd=qtd+1;  → qtd++;
    d=d+1;  → d++;
  }
  return qtd;
}
```



IMPORTANTE: Corpo deve modificar resultado da expressão

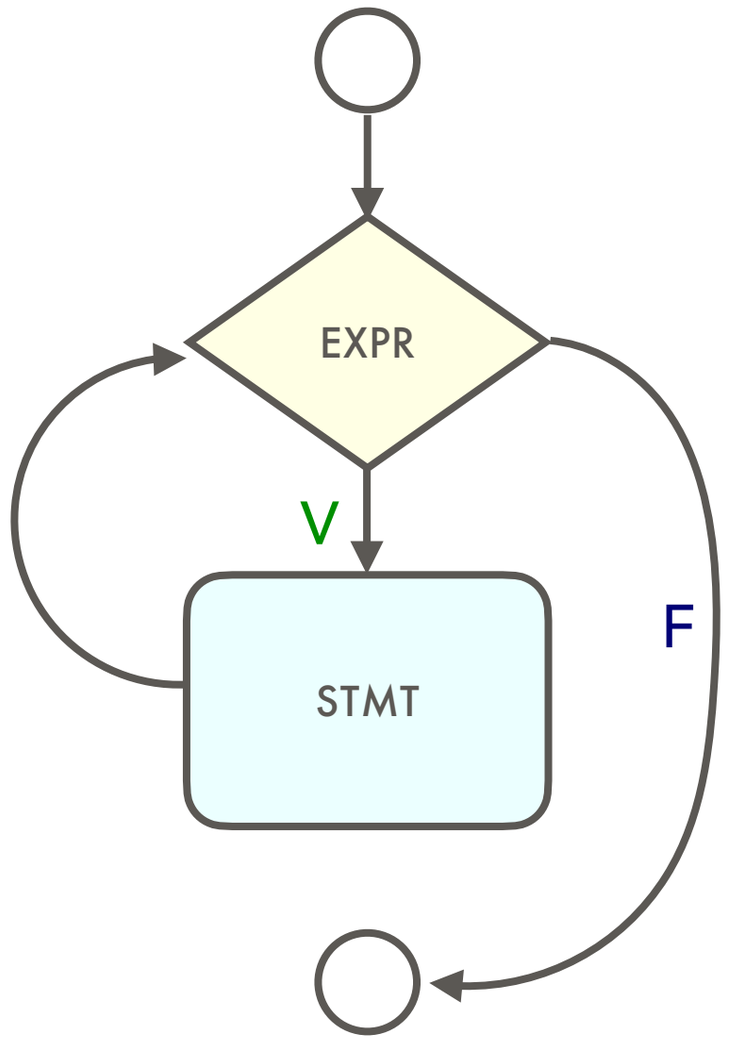
while

- Repete instruções enquanto teste (expressão) for verdadeiro
- Sintaxe:

```
while (expr) stmt;
```

- Exemplo: Contar divisores de n

```
int contar_divisores(int n){
  int d=1,qtd=1;
  while (d!=n){
    if ((n%d)==0)
      qtd++;
    d++;
  }
  return qtd;
}
```



IMPORTANTE: Corpo deve modificar resultado da expressão

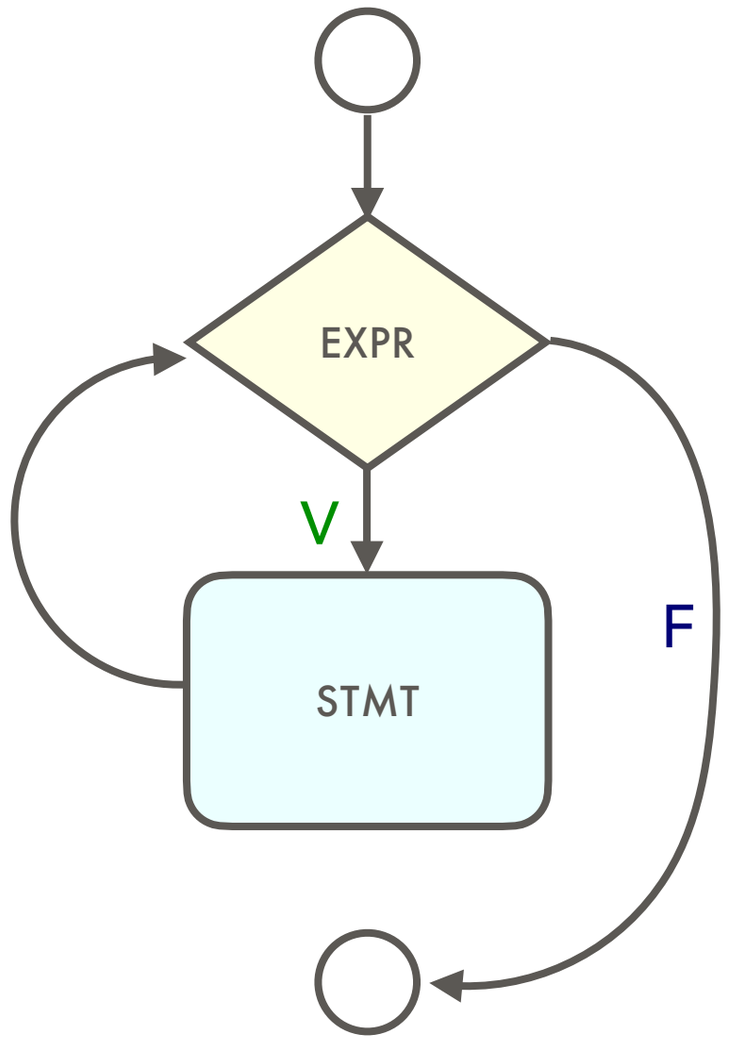
while

- Repete instruções enquanto teste (expressão) for verdadeiro
- Sintaxe:

```
while (expr) stmt;
```

- Exemplo: Contar divisores de n

```
int contar_divisores(int n){
  int d=1,qtd=1;
  while (d!=n){
    if ((n%d)==0)
      qtd++;
    d++;
  }
  return qtd;
}
```



IMPORTANTE: Corpo deve modificar resultado da expressão

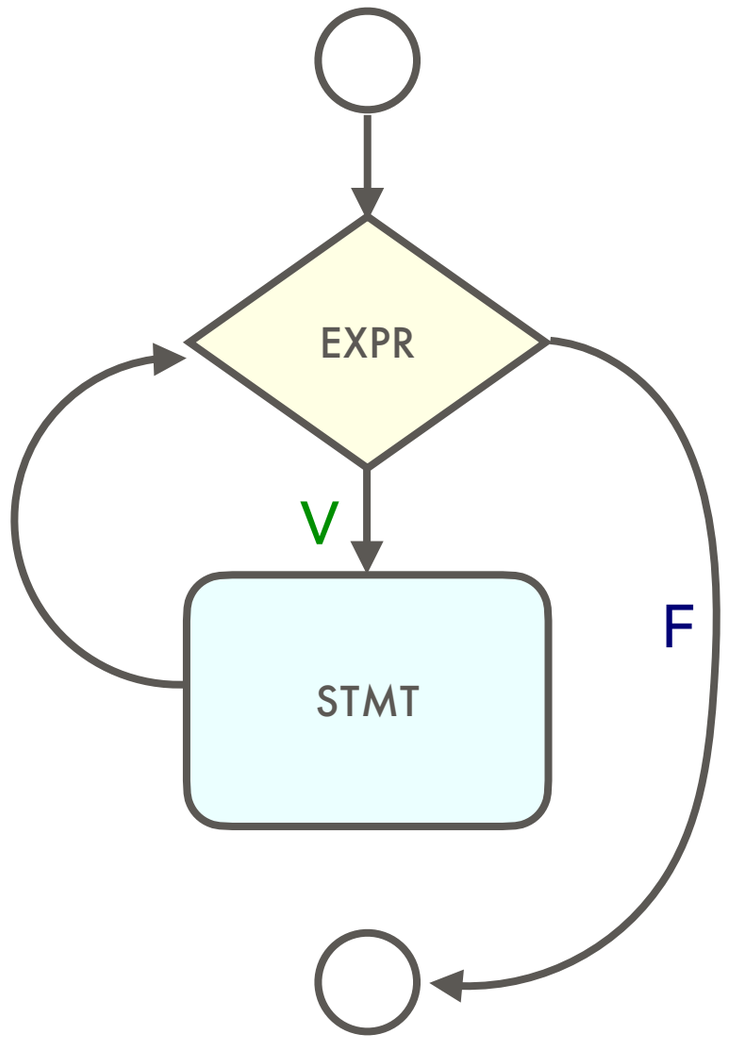
while

- Repete instruções enquanto teste (expressão) for verdadeiro
- Sintaxe:

```
while (expr) stmt;
```

- Exemplo: Contar divisores de n

```
int contar_divisores(int n){
  int d=1,qtd=1;
  while (d!=n){
    if ((n%d)==0)
      qtd++;
    d++;
  }
  return qtd;
}
```



IMPORTANTE: Corpo deve modificar resultado da expressão

while

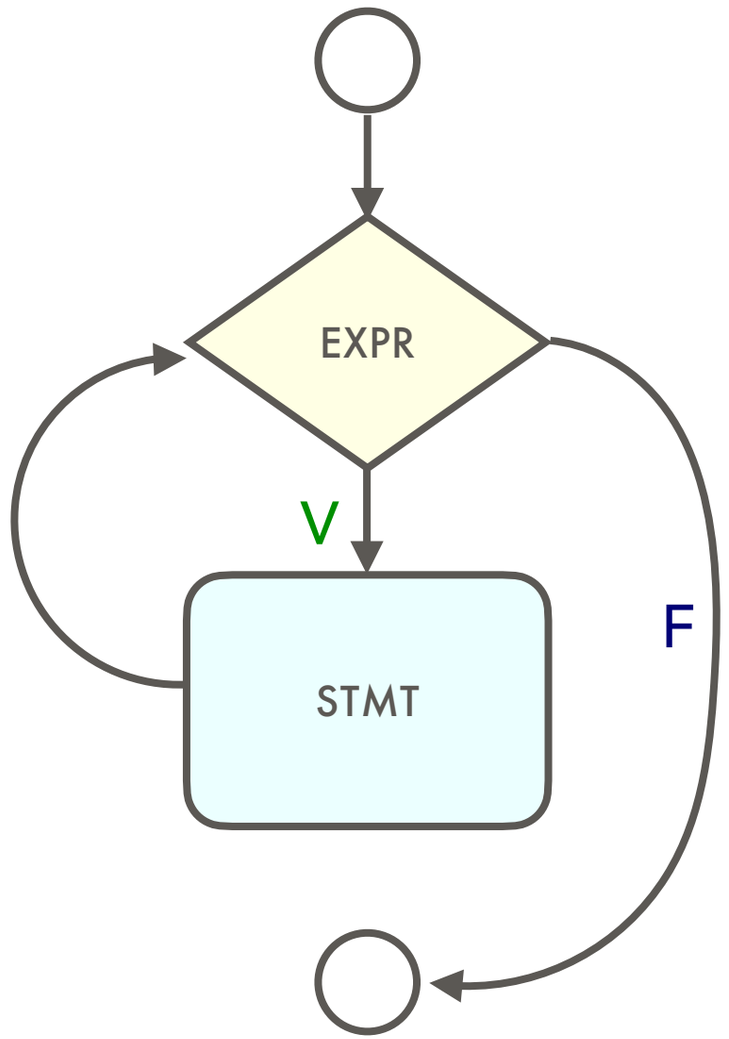
- Repete instruções enquanto teste (expressão) for verdadeiro
- Sintaxe:

```
while (expr) stmt;
```

- Exemplo: Contar divisores de n

```
int contar_divisores(int n){
  int d=1,qtd=1;
  while (d!=n)
    if ((n%d++)==0)
      qtd++;


  return qtd;
}
```



IMPORTANTE: Corpo deve modificar resultado da expressão

do..while

do..while

- Repete instruções enquanto teste (expressão) for verdadeiro
  - Teste é feito após primeira execução
- Sintaxe:

```
do stmt while (expr);
```

do..while

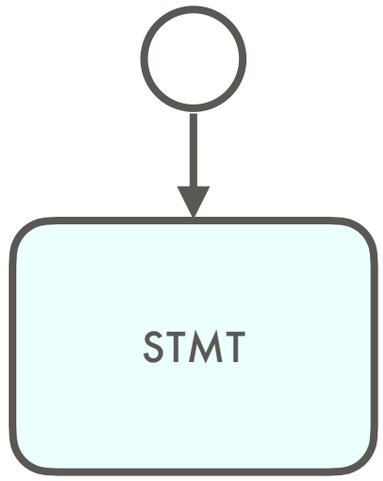
- Repete instruções enquanto teste (expressão) for verdadeiro
  - Teste é feito após primeira execução
- Sintaxe:

```
do stmt while (expr);
```

do..while

- Repete instruções enquanto teste (expressão) for verdadeiro
  - Teste é feito após primeira execução
- Sintaxe:

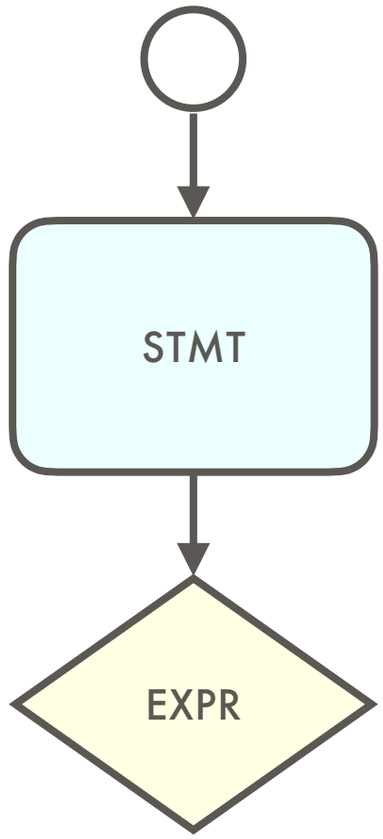
```
do stmt while (expr);
```



do..while

- Repete instruções enquanto teste (expressão) for verdadeiro
  - Teste é feito após primeira execução
- Sintaxe:

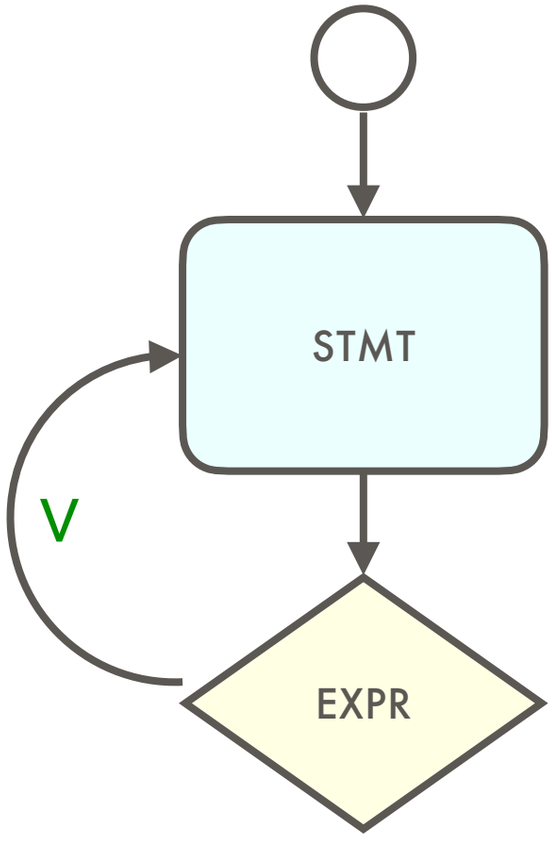
```
do stmt while (expr);
```



do..while

- Repete instruções enquanto teste (expressão) for verdadeiro
  - Teste é feito após primeira execução
- Sintaxe:

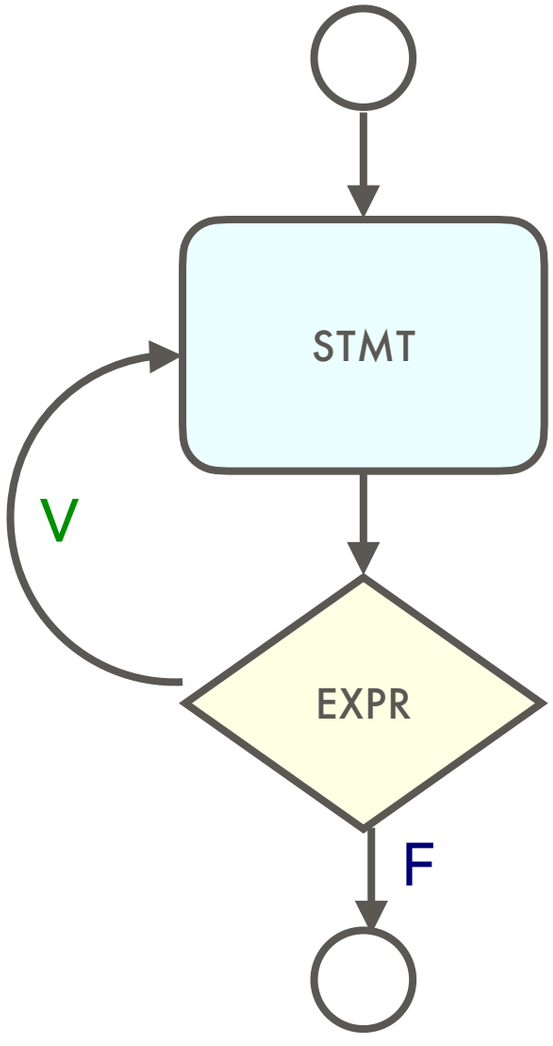
```
do stmt while (expr);
```



do..while

- Repete instruções enquanto teste (expressão) for verdadeiro
  - Teste é feito após primeira execução
- Sintaxe:

```
do stmt while (expr);
```



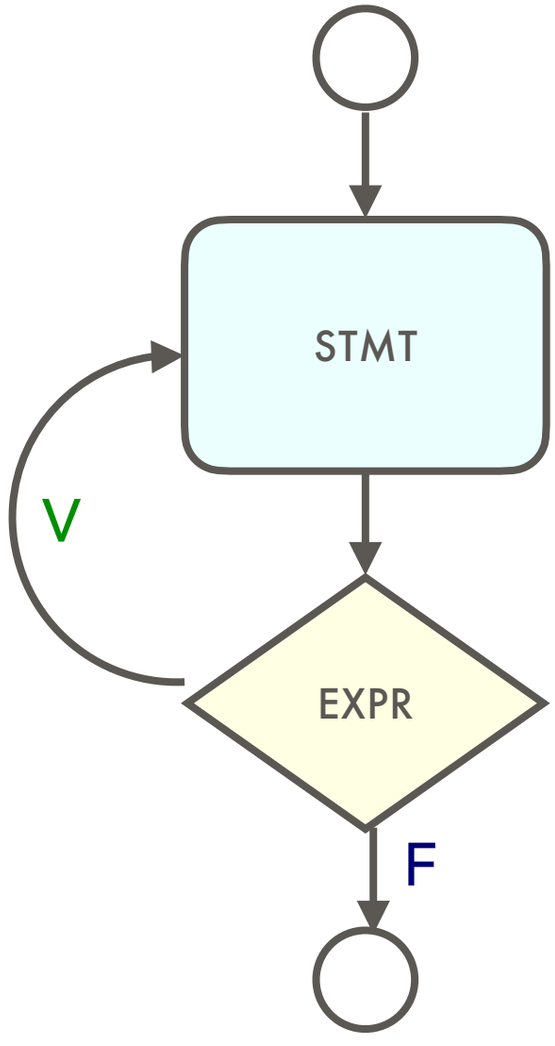
do..while

- Repete instruções enquanto teste (expressão) for verdadeiro
  - Teste é feito após primeira execução
- Sintaxe:

```
do stmt while (expr);
```

- Exemplo: Contar divisores de n

```
int contar_divisores(int n){
  int qtd=1,d=1;
  do {
    if ((n%d++)==0)
      qtd++;
  } while (d!=n);
  return qtd;
}
```



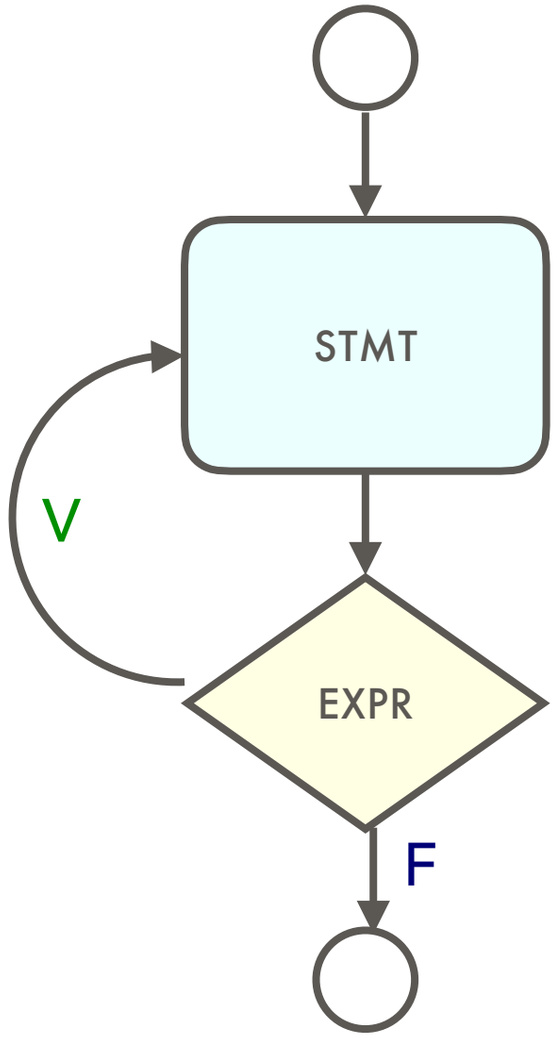
do..while

- Repete instruções enquanto teste (expressão) for verdadeiro
  - Teste é feito após primeira execução
- Sintaxe:

```
do stmt while (expr);
```

- Exemplo: Contar divisores de n

```
int contar_divisores(int n){
  int qtd=1,d=1;
  do {
    if ((n%d++)==0)
      qtd++;
  } while (d!=n);
  return qtd;
}
```



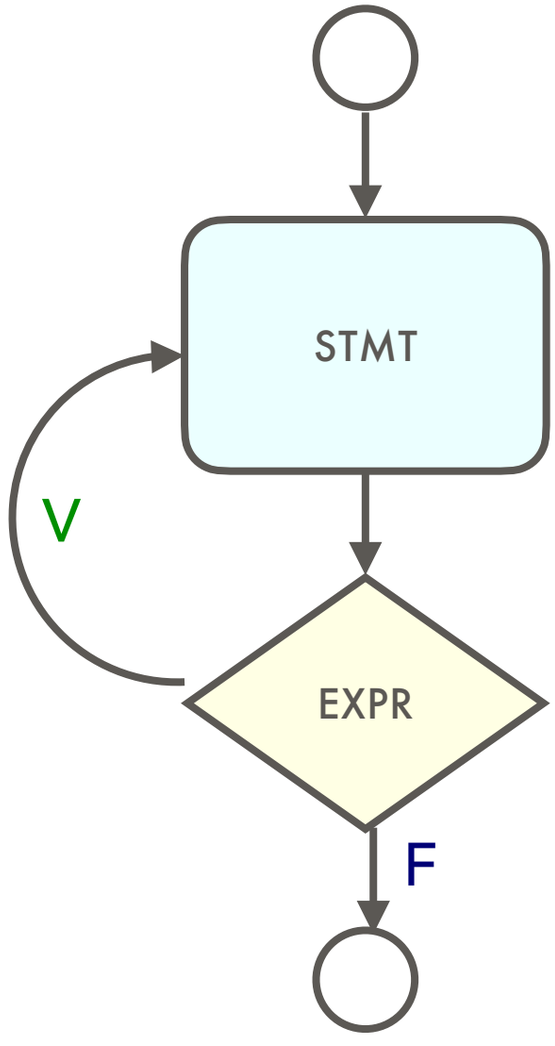
do..while

- Repete instruções enquanto teste (expressão) for verdadeiro
  - Teste é feito após primeira execução
- Sintaxe:

```
do stmt while (expr);
```

- Exemplo: Contar divisores de n

```
int contar_divisores(int n){
  int qtd=1,d=1;
  do {
    if ((n%d++)==0)
      qtd++;
  } while (d!=n);
  return qtd;
}
```



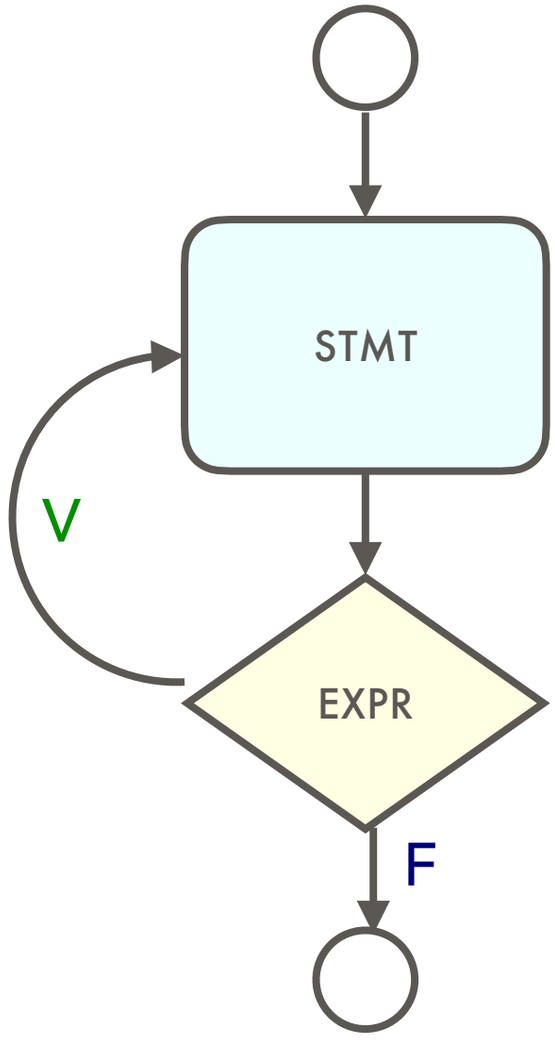
do..while

- Repete instruções enquanto teste (expressão) for verdadeiro
  - Teste é feito após primeira execução
- Sintaxe:

```
do stmt while (expr);
```

- Exemplo: Contar divisores de n

```
int contar_divisores(int n){
  int qtd=1,d=1;
  do {
    if ((n%d++)==0)
      qtd++;
  } while (d!=n);
  return qtd;
}
```



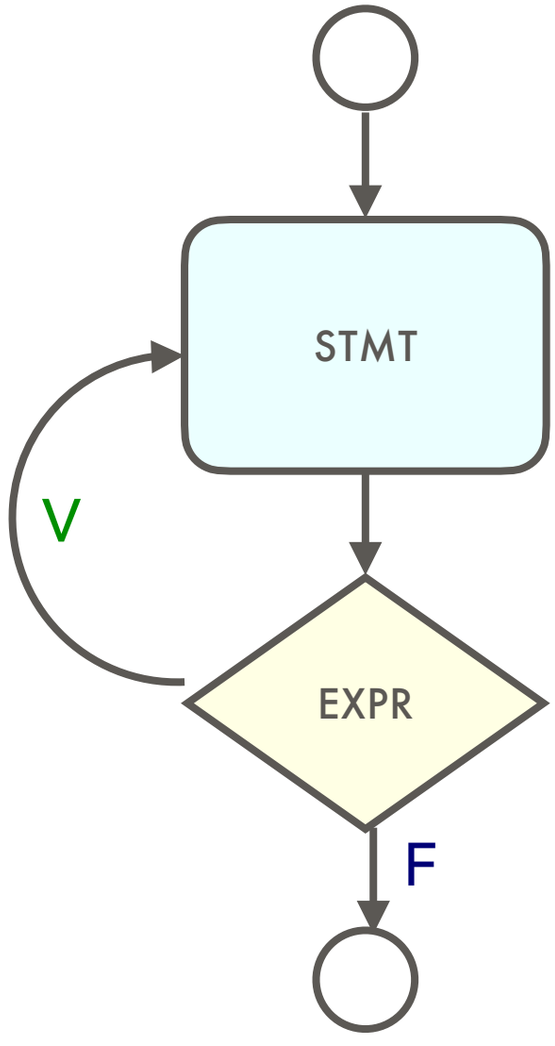
do..while

- Repete instruções enquanto teste (expressão) for verdadeiro
  - Teste é feito após primeira execução
- Sintaxe:

```
do stmt while (expr);
```

- Exemplo: Contar divisores de n

```
int contar_divisores(int n){
  int qtd=1,d=1;
  do
    if ((n%d++)==0)
      qtd++;
    while (d!=n);
  return qtd;
}
```



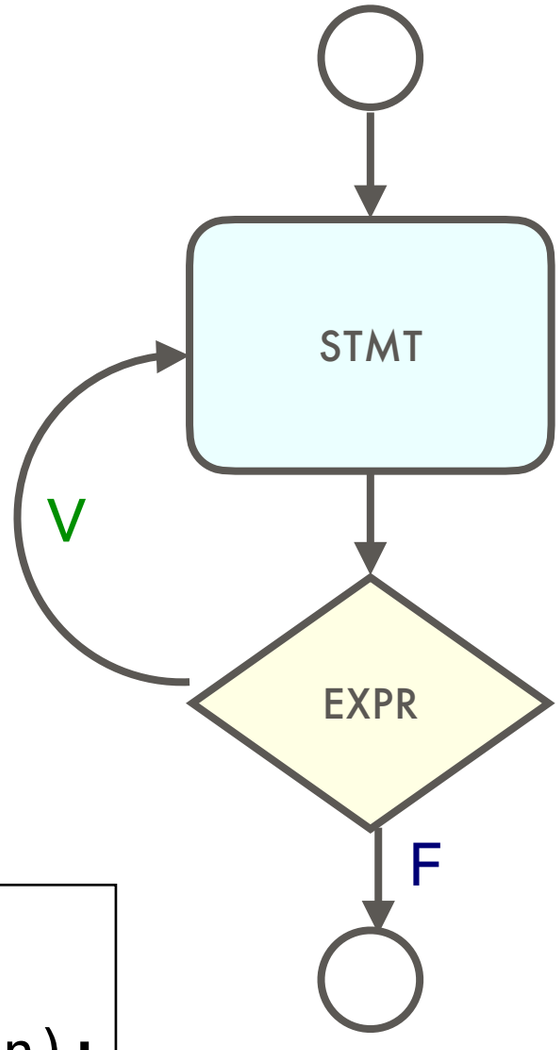
do..while

- Repete instruções enquanto teste (expressão) for verdadeiro
  - Teste é feito após primeira execução
- Sintaxe:

```
do stmt while (expr);
```

- Exemplo: Contar divisores de n

```
int contar_divisores(int n){
  int qtd=1,d=1;
  do
    if ((n%d++)==0)
      qtd++;
    while (d!=n);
  return qtd;
}
```




```
int contar_divisores2(int n){
  int qtd=1,d=1;
  do if ((n%d++)==0) qtd++; while (d!=n);
  return qtd;
}
```

for

for

- Semelhante ao `while`
  - Código de fácil leitura
- Sintaxe

```
for ( init ; expr ; iter ) stmt;
```

for

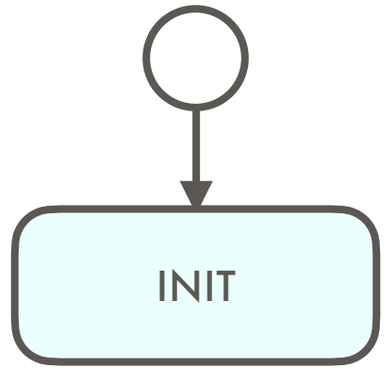
- Semelhante ao `while`
  - Código de fácil leitura
- Sintaxe

```
for ( init ; expr ; iter ) stmt;
```

for

- Semelhante ao `while`
  - Código de fácil leitura
- Sintaxe

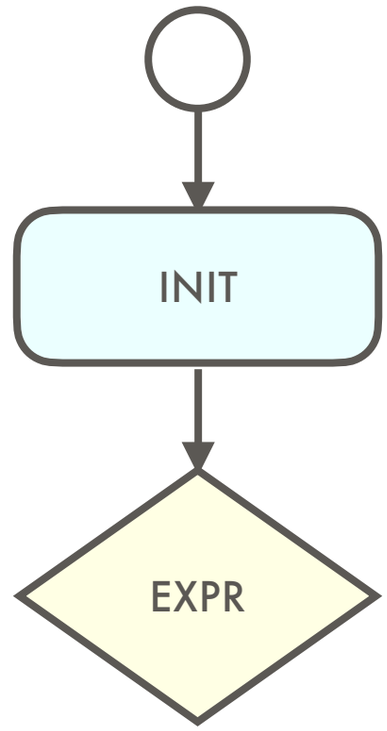
```
for ( init ; expr ; iter ) stmt;
```



for

- Semelhante ao `while`
  - Código de fácil leitura
- Sintaxe

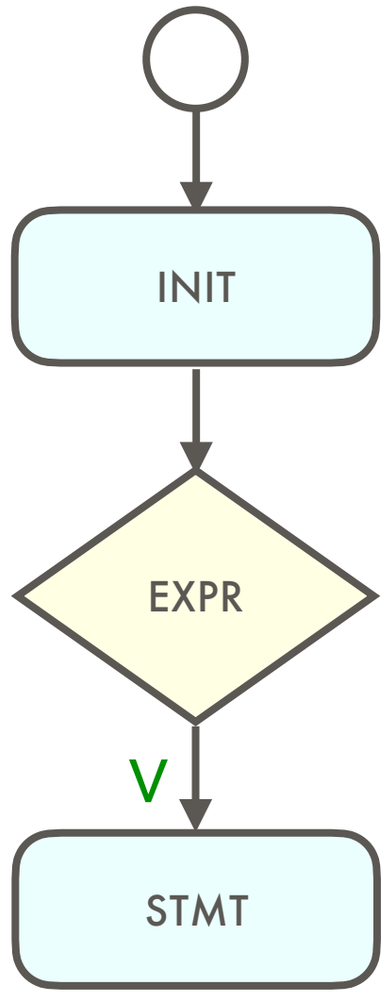
```
for ( init ; expr ; iter ) stmt;
```



for

- Semelhante ao `while`
  - Código de fácil leitura
- Sintaxe

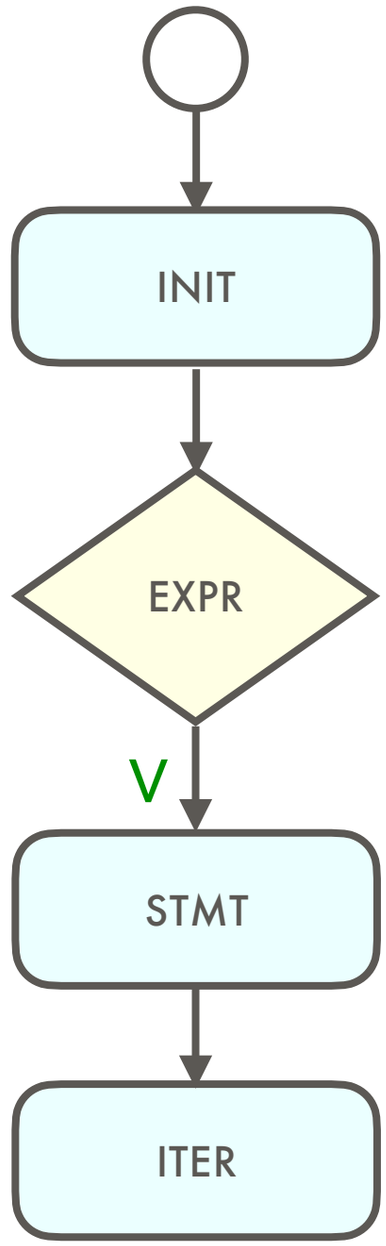
```
for ( init ; expr ; iter ) stmt;
```



for

- Semelhante ao `while`
  - Código de fácil leitura
- Sintaxe

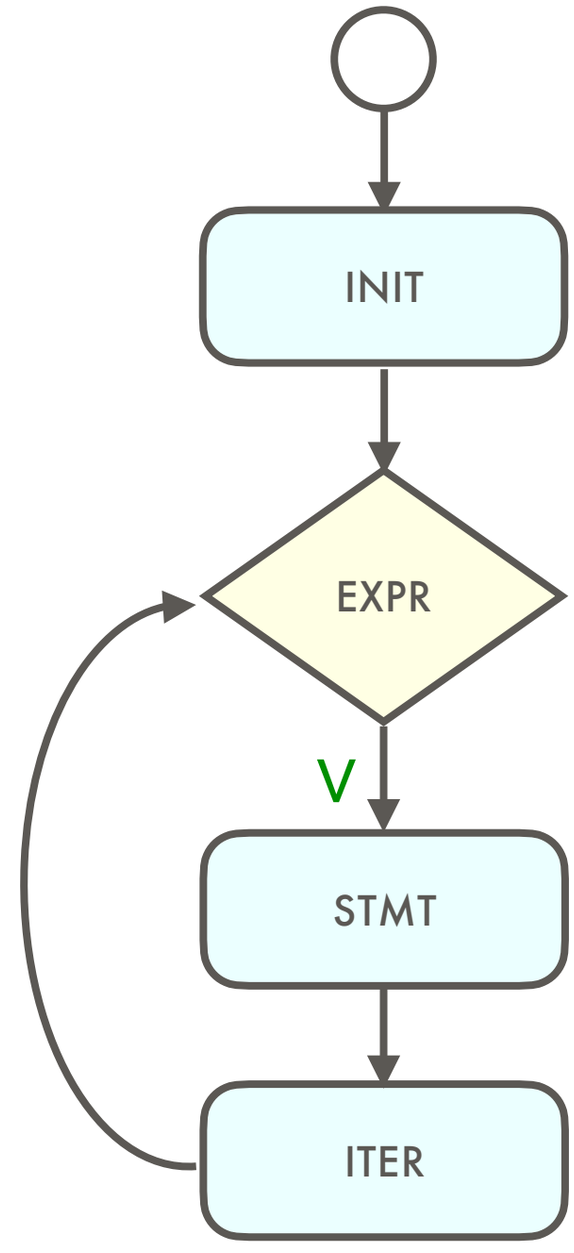
```
for ( init ; expr ; iter ) stmt;
```



for

- Semelhante ao `while`
  - Código de fácil leitura
- Sintaxe

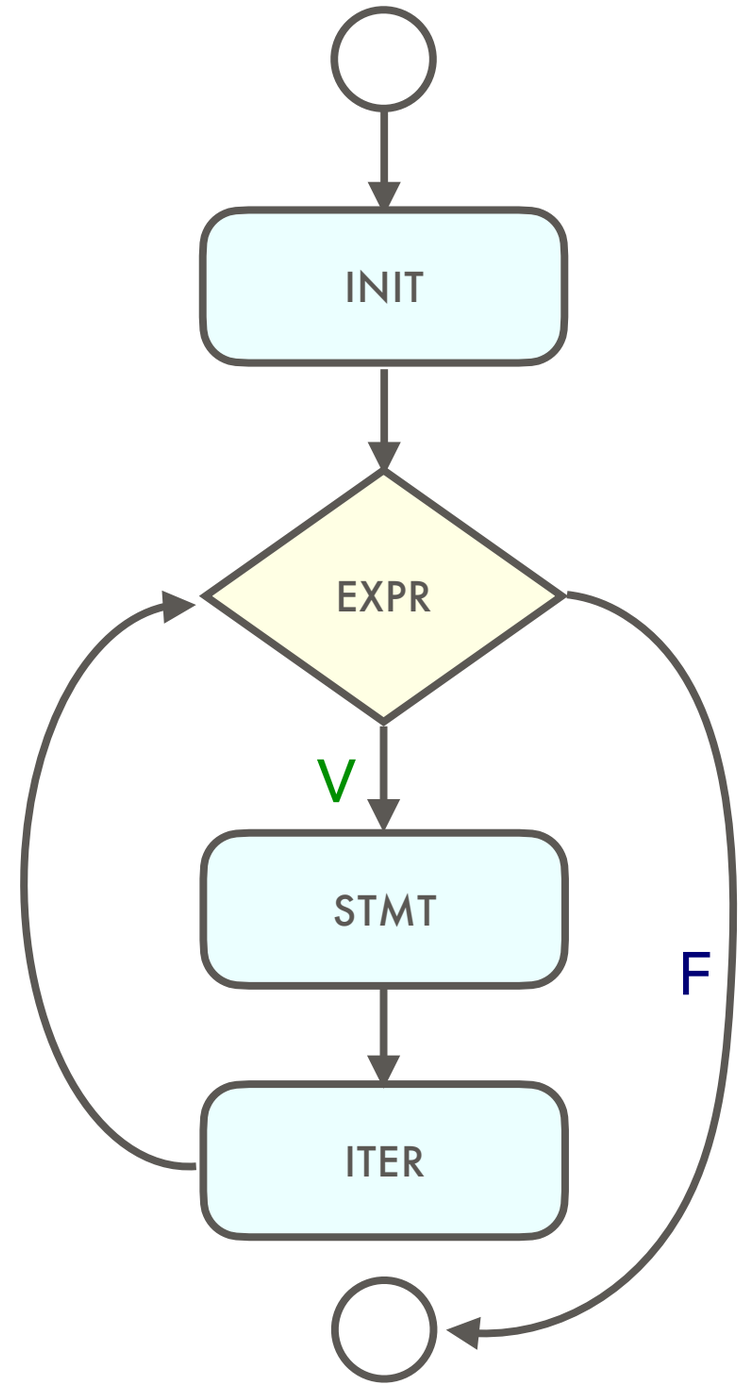
```
for ( init ; expr ; iter ) stmt;
```



for

- Semelhante ao `while`
  - Código de fácil leitura
- Sintaxe

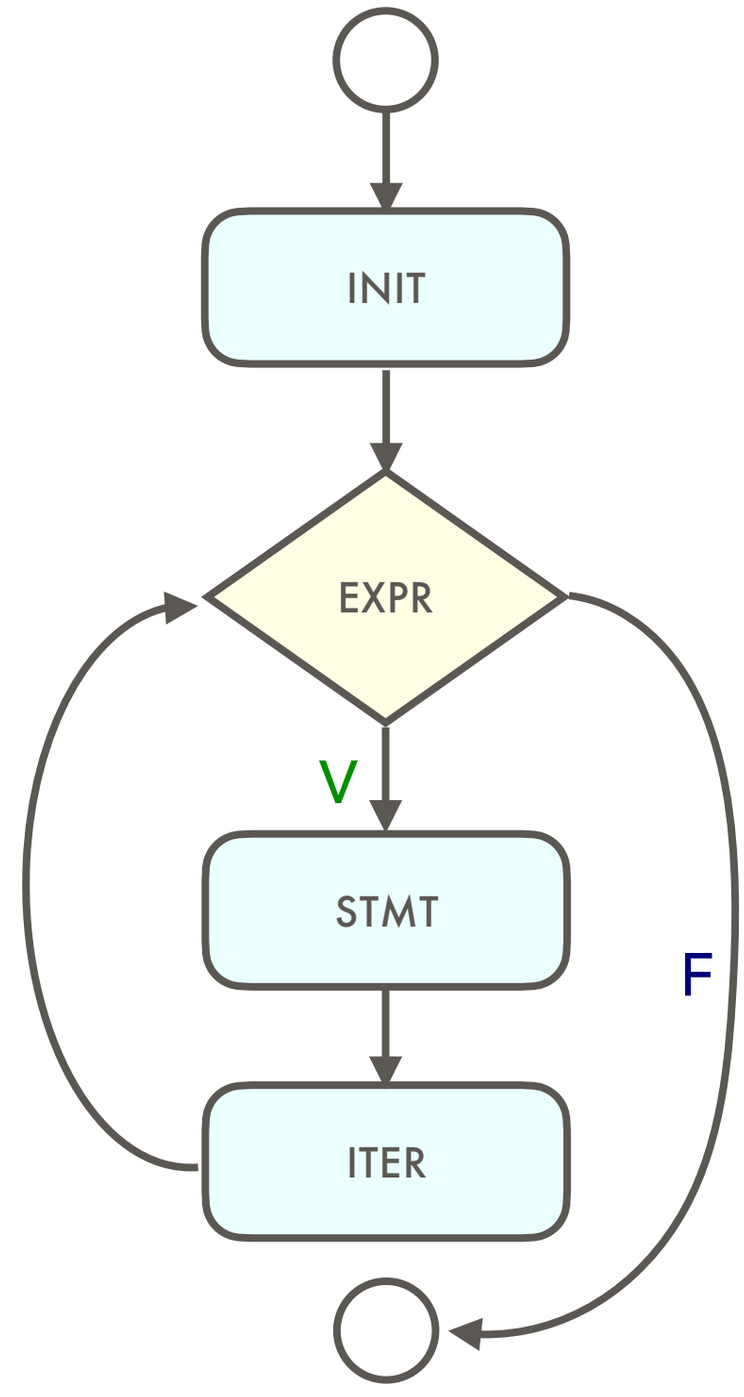
```
for ( init ; expr ; iter ) stmt;
```



for

- Semelhante ao `while`
  - Código de fácil leitura
- Sintaxe

```
for ( init ; expr ; iter ) stmt;
```



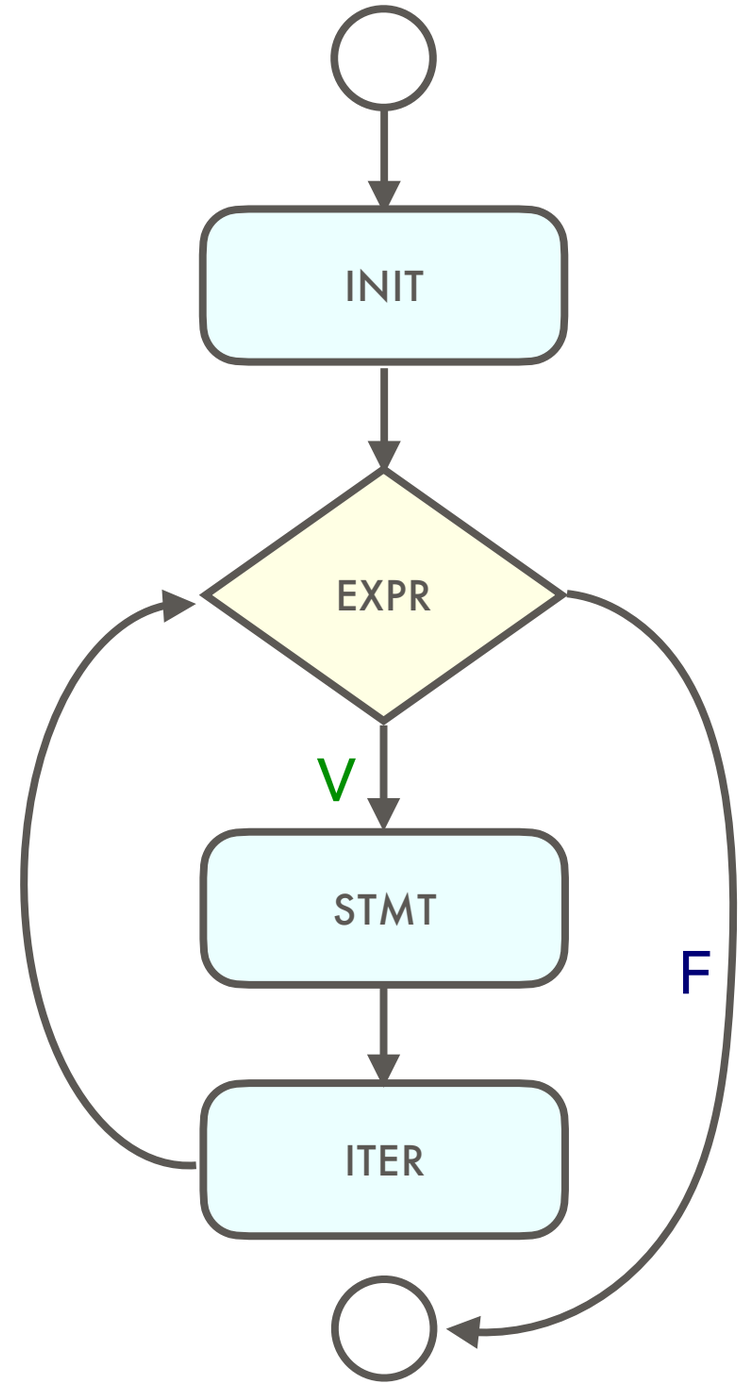
for

- Semelhante ao `while`
  - Código de fácil leitura
- Sintaxe

```
for ( init ; expr ; iter ) stmt;
```

- Exemplo

```
int contar_divisores(int n){
  int i,qtd=1;
  for (i=1 ; i<n ; ++i){
    if ((n%i)==0){
      qtd++;
    }
  }
  return qtd;
}
```



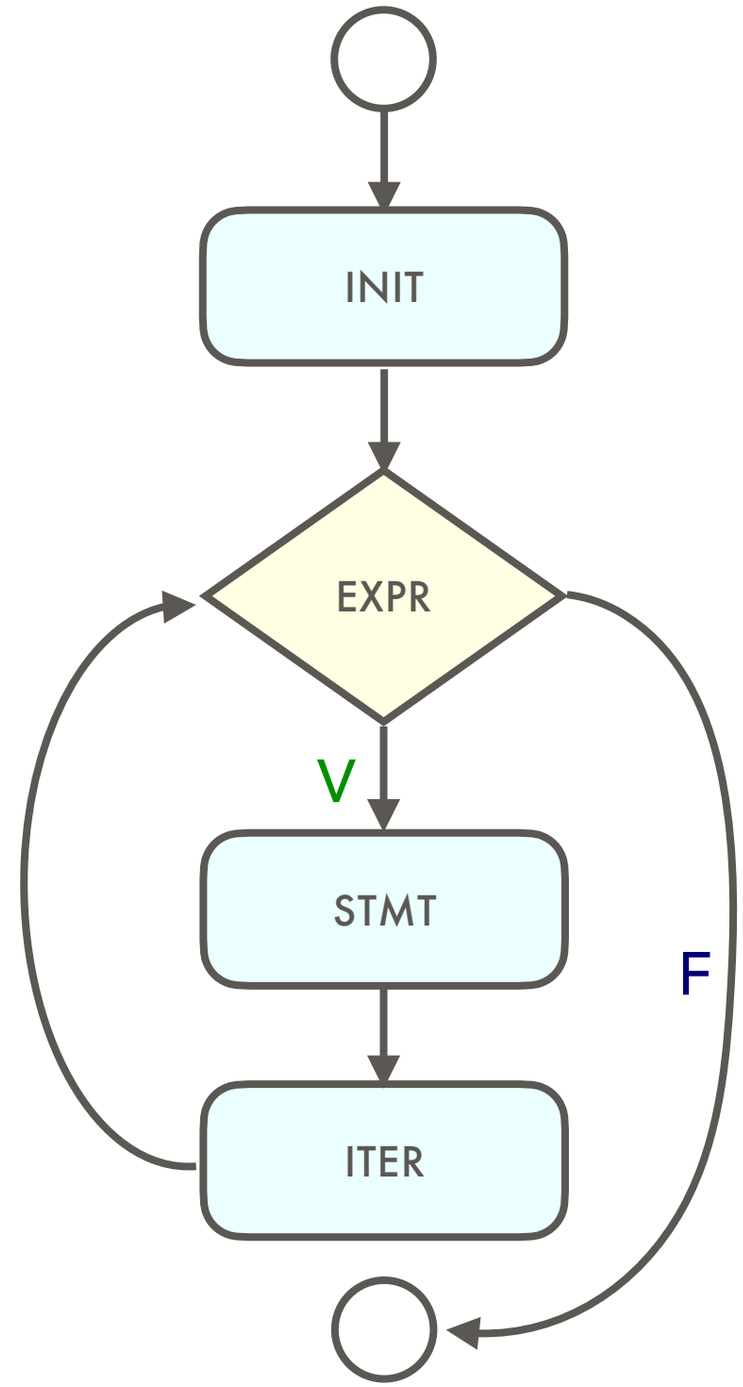
for

- Semelhante ao `while`
  - Código de fácil leitura
- Sintaxe

```
for ( init ; expr ; iter ) stmt;
```

- Exemplo

```
int contar_divisores(int n){
  int i,qtd=1;
  for (i=1 ; i<n ; ++i){
    if ((n%i)==0){
      qtd++;
    }
  }
  return qtd;
}
```



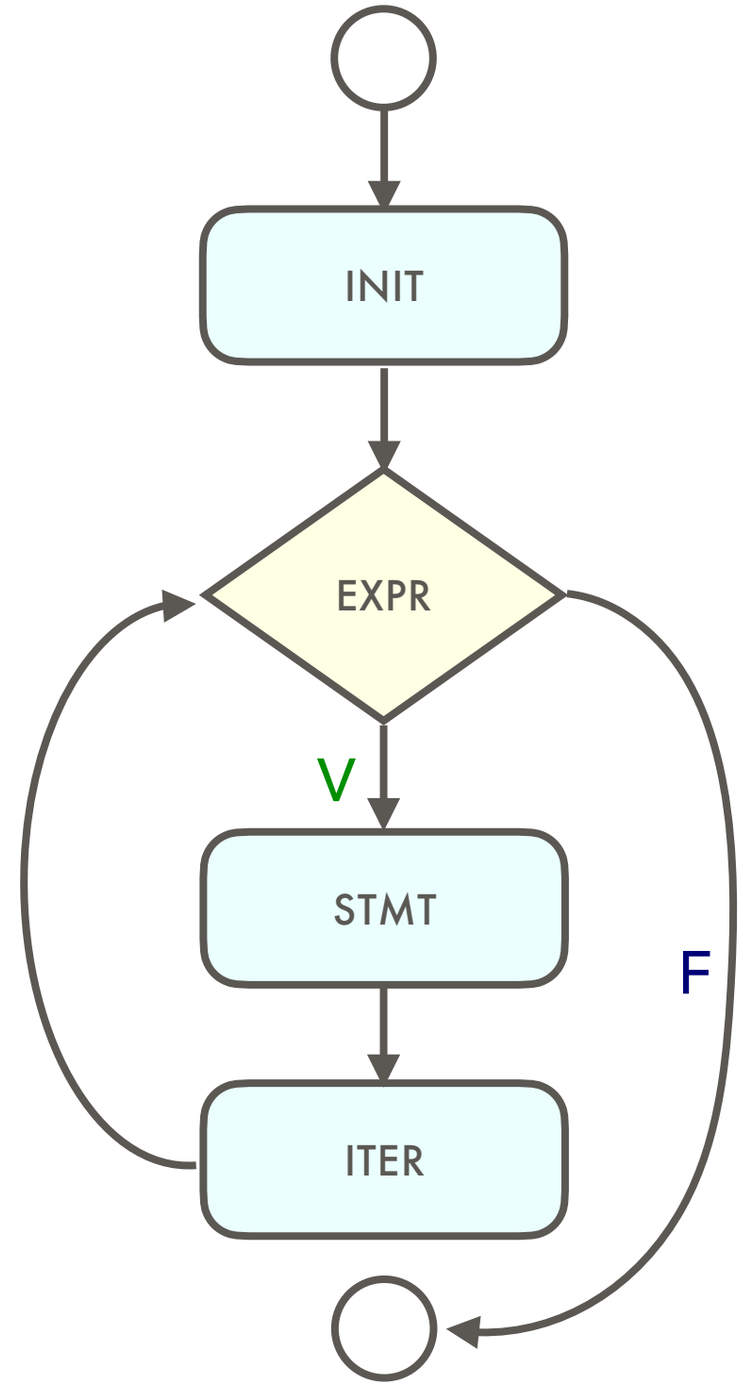
for

- Semelhante ao `while`
  - Código de fácil leitura
- Sintaxe

```
for ( init ; expr ; iter ) stmt;
```

- Exemplo

```
int contar_divisores(int n){
  int i,qtd=1;
  for (i=1 ; i<n ; ++i){
    if ((n%i)==0){
      qtd++;
    }
  }
  return qtd;
}
```



for

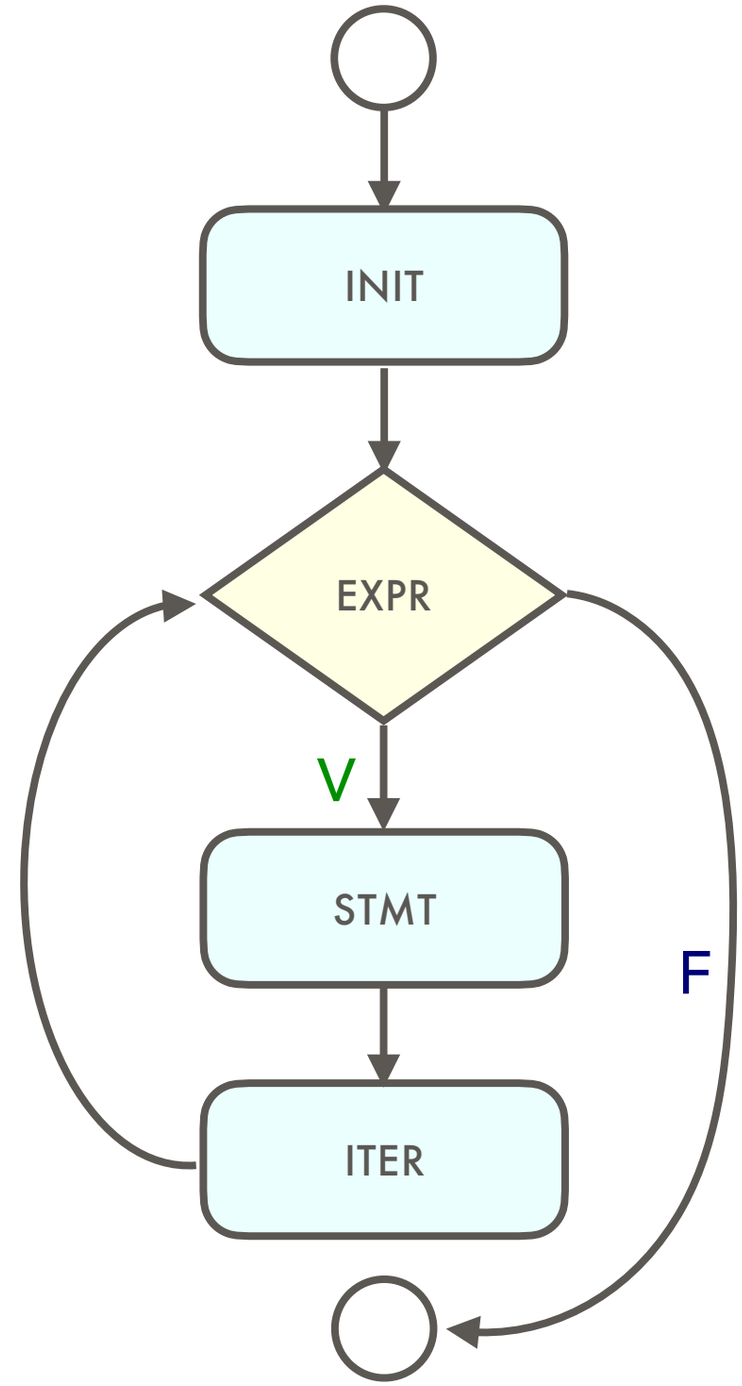
- Semelhante ao `while`
  - Código de fácil leitura
- Sintaxe

```
for ( init ; expr ; iter ) stmt;
```

- Exemplo

```
int contar_divisores(int n){
  int i,qtd=1;
  for (i=1 ; i<n ; ++i){
    if ((n%i)==0){
      qtd++;
    }
  }
  return qtd;
}
```

stmt



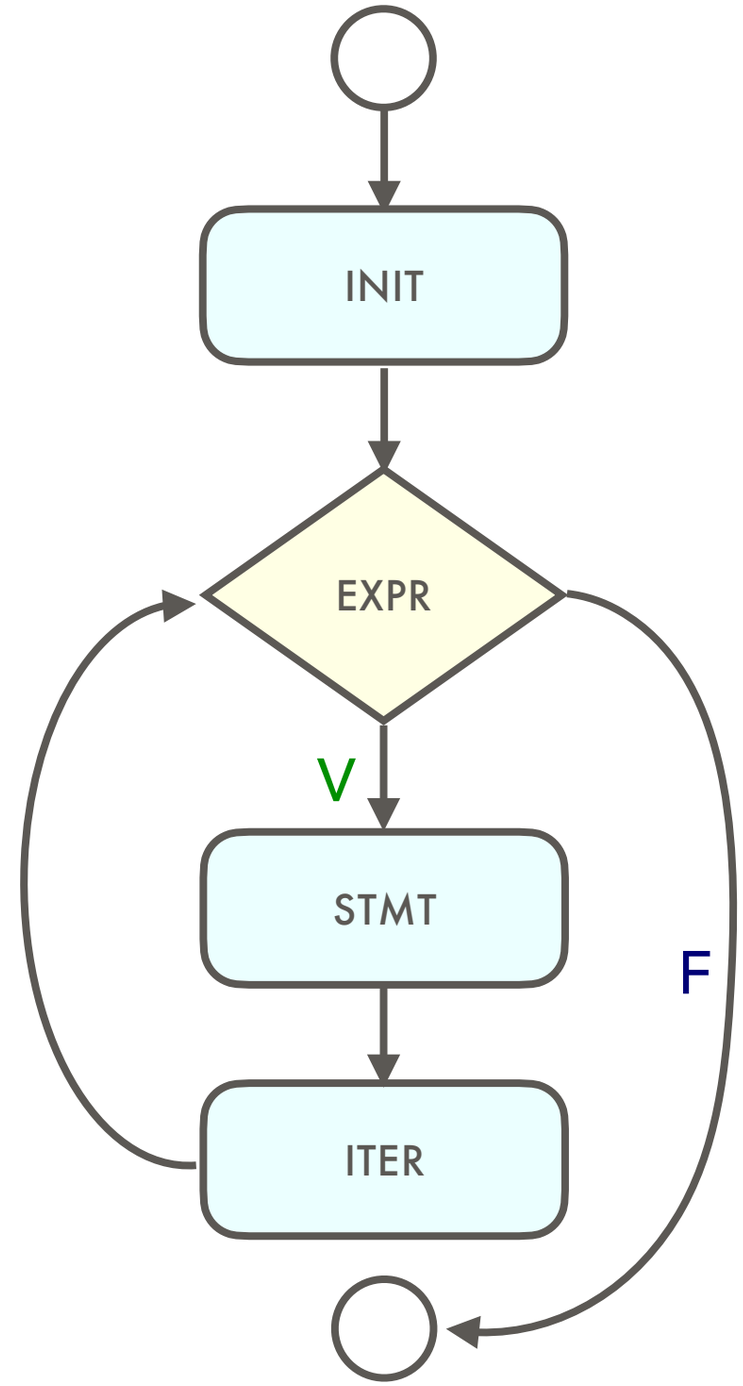
for

- Semelhante ao `while`
  - Código de fácil leitura
- Sintaxe

```
for ( init ; expr ; iter ) stmt;
```

- Exemplo

```
int contar_divisores(int n){
  int i,qtd=1;
  for (i=1 ; i<n ; ++i){
    if ((n%i)==0){
      qtd++;
    }
  }
  return qtd;
}
```



continue

- Pula código restante do laço
  - Volta para avaliação de expressão

continue

- Pula código restante do laço
  - Volta para avaliação de expressão

```
while (expr){

  /* Trecho de código */

  continue;

  /* Trecho de código */

}
```

continue

- Pula código restante do laço
  - Volta para avaliação de expressão

```
while (expr){
  /* Trecho de código */
  continue;
  /* Trecho de código */
}
```

continue

- Pula código restante do laço
  - Volta para avaliação de expressão

```
while (expr){

  /* Trecho de código */

  continue;

  /* Trecho de código */

}
```

continue

- Pula código restante do laço
  - Volta para avaliação de expressão

```
while (expr){

  /* Trecho de código */

  if (/*devo continuar?*/)
    continue;

  /* Trecho de código */

}
```

break

- Termina o laço
  - Não avalia expressão

break

- Termina o laço
  - Não avalia expressão

```
while (expr){

    /* Trecho de código */

    break;

    /* Trecho de código */

}
```

break

- Termina o laço
  - Não avalia expressão

```
while (expr){

    /* Trecho de código */

    break;

    /* Trecho de código */

}

/* Trecho de código */
```

break

- Termina o laço
  - Não avalia expressão

```
while (expr){

    /* Trecho de código */

    break;

    /* Trecho de código */

}
```

break

- Termina o laço
  - Não avalia expressão

```
while (expr){

    /* Trecho de código */

    if (/*devo terminar o laço?*/)
      break;

    /* Trecho de código */

}
```

Prática

- Dúvidas?

Programação

C++

Aula 02

Jorgiano Vidal