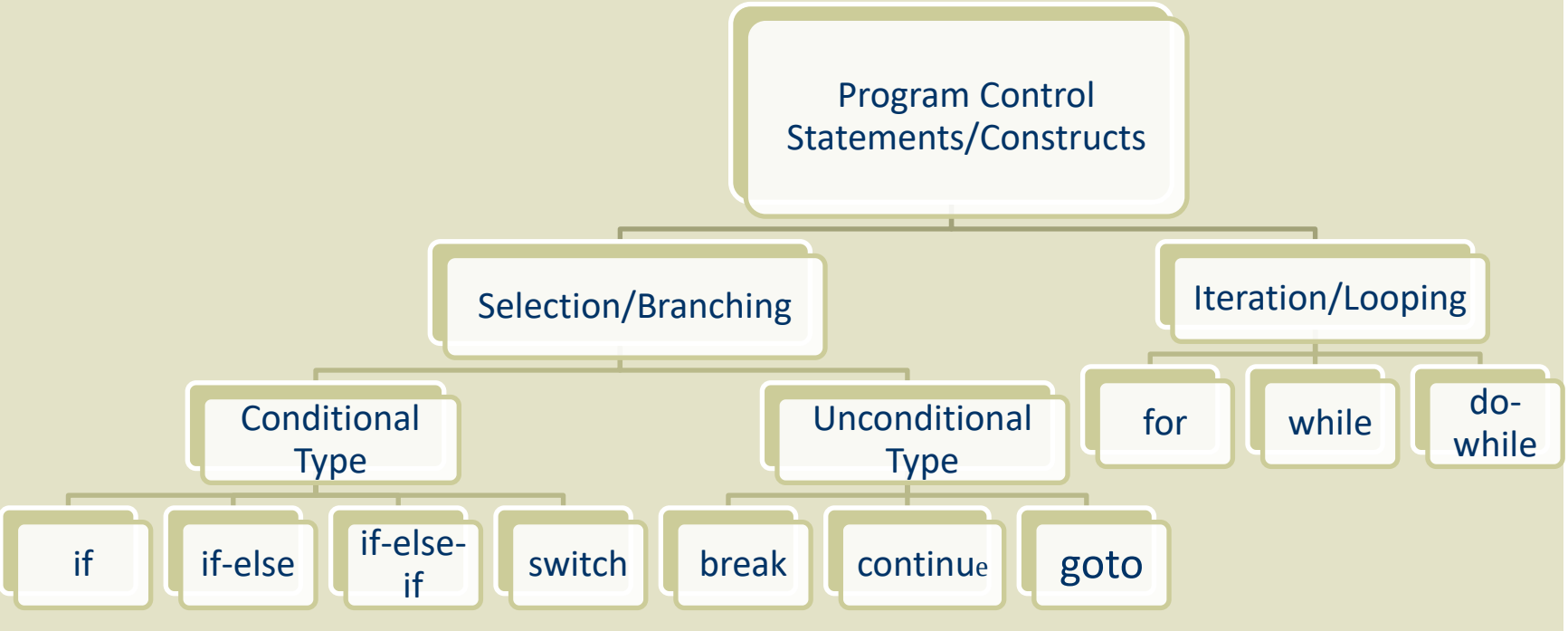
Fundamentos C/C++

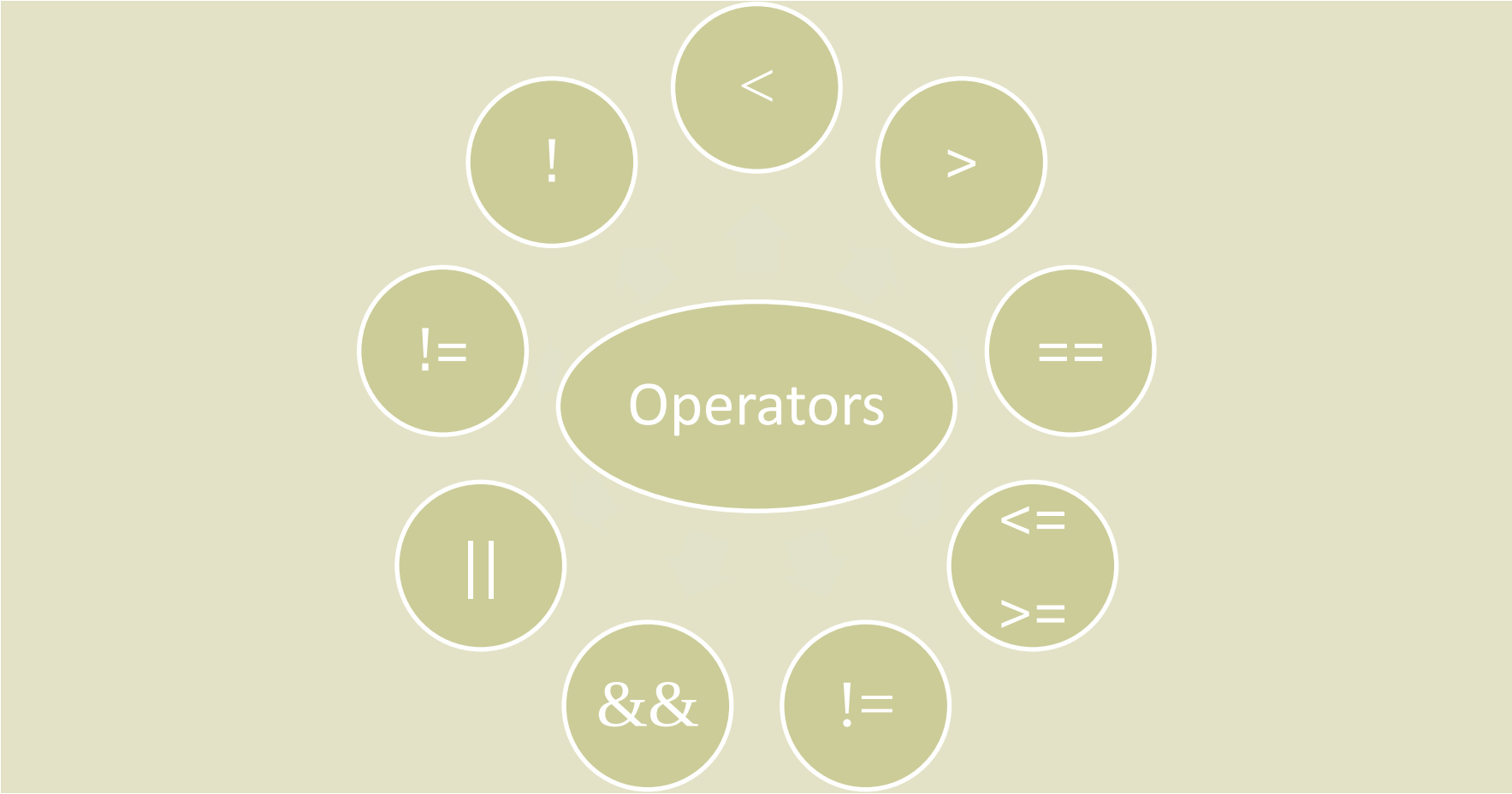
- Estruturas de controlo
- Operador ternário
- Enumerações
- Estruturas de control
- Operador ternário
- Estruturas iterativas
- Debugging
- Arrays/Vetores
- Funções
- Escopo de variáveis
- Procedimentos
- Tipos de dados *Boolean*
- Alocação dinâmica de memória
- Structs

Hugo Dias
dias.hugo@gmail.com

Estruturas de controlo



Estruturas de controlo

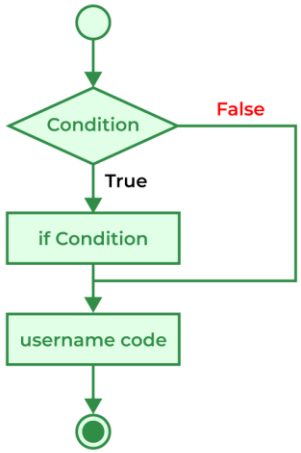


Estruturas de controlo - *if*

- Estrutura de controlo que executa um bloco de instruções se uma **determinada** condição é verdadeira

- Síntaxe:

```
if (<test>) {
    <statement(s)> ;
}
```



- Exemplo:

```
int code = 1315;
if (code >= 1300) {
    printf("codigo valido!");
}
```

Estruturas de controlo - *if*

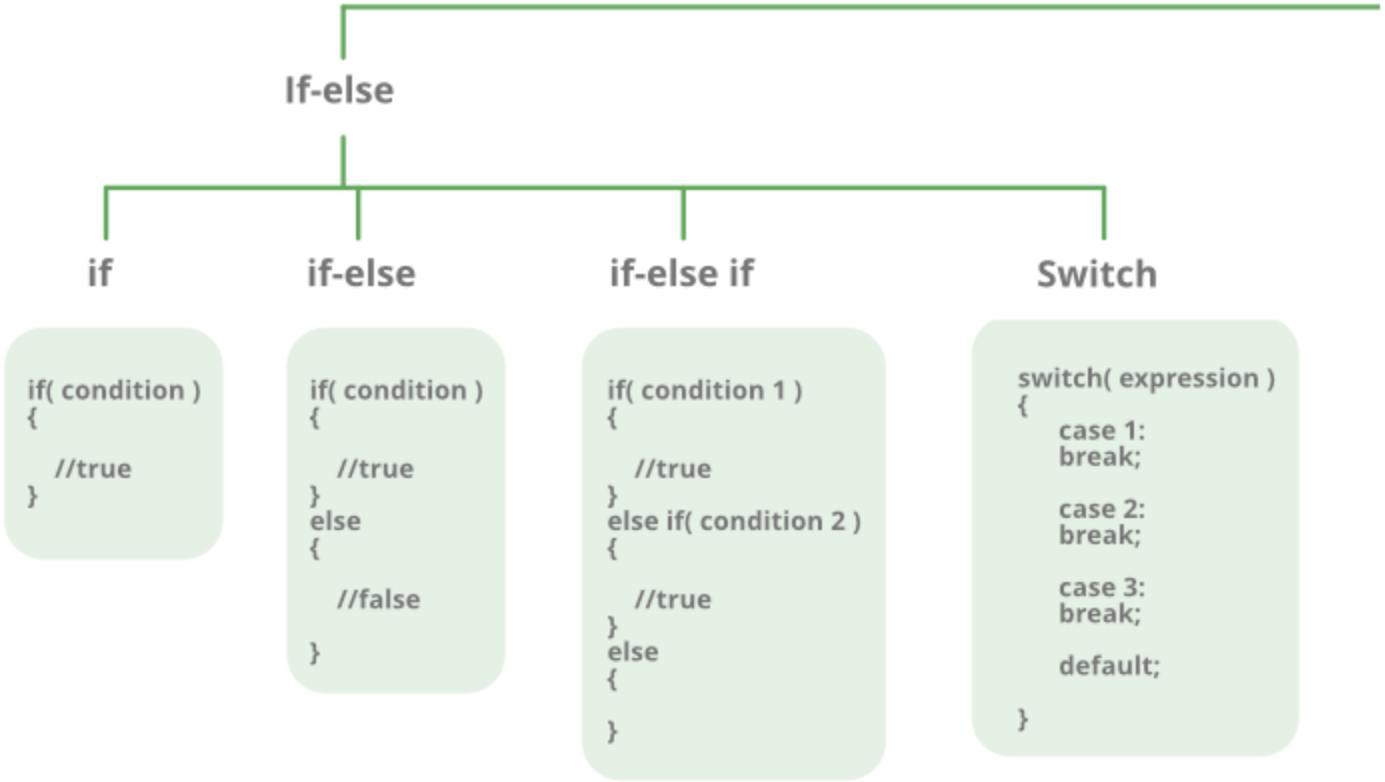


Tabela ASCII

```c
#include <stdio.h>
int main() {
  char ch;

  printf("Introduza um caracter:");
  scanf("%c", &ch );

  if( ch >='a' && ch <='z' ) {
    printf("O caracter pertence ao alfabeto\n");
  }

  return 0;
}
```

```
cook@pop-os:~$ ascii -d
   0 NUL    16 DLE    32      48 0    64 @    80 P    96 `   112 p
   1 SOH    17 DC1    33 !    49 1    65 A    81 Q    97 a   113 q
   2 STX    18 DC2    34 "    50 2    66 B    82 R    98 b   114 r
   3 ETX    19 DC3    35 #    51 3    67 C    83 S    99 c   115 s
   4 EOT    20 DC4    36 $    52 4    68 D    84 T   100 d   116 t
   5 ENQ    21 NAK    37 %    53 5    69 E    85 U   101 e   117 u
   6 ACK    22 SYN    38 &    54 6    70 F    86 V   102 f   118 v
   7 BEL    23 ETB    39 '    55 7    71 G    87 W   103 g   119 w
   8 BS     24 CAN    40 (    56 8    72 H    88 X   104 h   120 x
   9 HT     25 EM     41 )    57 9    73 I    89 Y   105 i   121 y
  10 LF     26 SUB    42 *    58 :    74 J    90 Z   106 j   122 z
  11 VT     27 ESC    43 +    59 ;    75 K    91 [   107 k   123 {
  12 FF     28 FS     44 ,    60 <    76 L    92 \   108 l   124 |
  13 CR     29 GS     45 -    61 =    77 M    93 ]   109 m   125 }
  14 SO     30 RS     46 .    62 >    78 N    94 ^   110 n   126 ~
  15 SI     31 US     47 /    63 ?    79 O    95 _   111 o   127 DEL
```

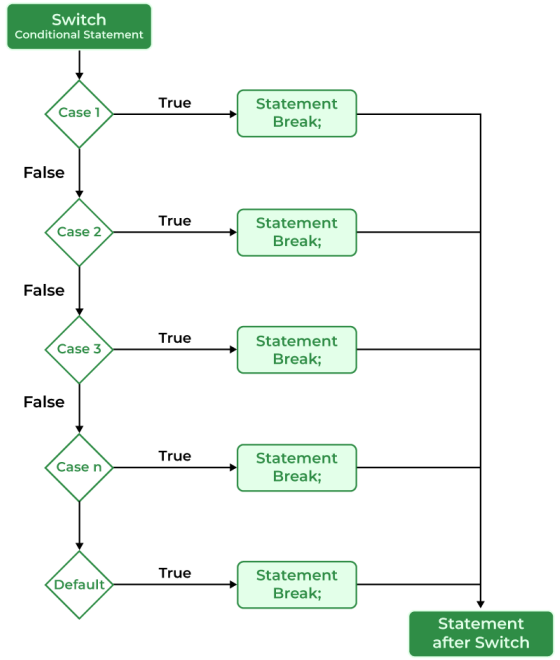
```c
#include<stdio.h>
void main ()
{

  int num;
  printf ("=====This Program Converts ASCII to Alphabet!=====\n");
  printf ("Enter ASCII: ");
  scanf ("%d", &num);
  printf("%d is ASCII value of '%c'", num, (char)num );
}
```

```c
#include<stdio.h>

void main ()
{

  char alphabet;
  printf ("=====This Program Converts Alphabet to ASCII code!=====\n");
  printf ("Enter Alphabet: ");
  scanf ("%c", &alphabet);
  printf("ASCII value of '%c' is %d", alphabet, (char)alphabet );
}
```

Estruturas de controlo - *switch*

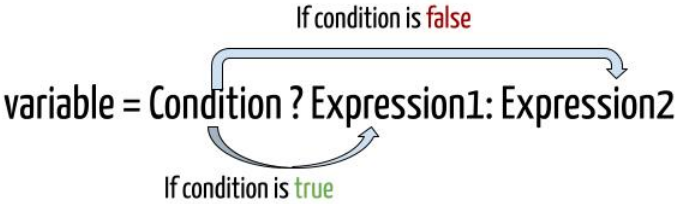
- Estrutura de controlo que testa uma expressão e compara-a a múltiplas opções e executa o bloco da opção correta

```c
#include <stdio.h>
    int main() {
        int num = 8;
    switch (num) {
            case 7:
                printf("Value is 7");
                break;
            case 8:
                printf("Value is 8");
                break;
            case 9:
                printf("Value is 9");
                break;
            default:
                printf("Out of range");
                break;
        }
        return 0;
    }
```



Operador ternário

- Operador condicional
- Forma abreviada de substituir o *if...else*



if...else

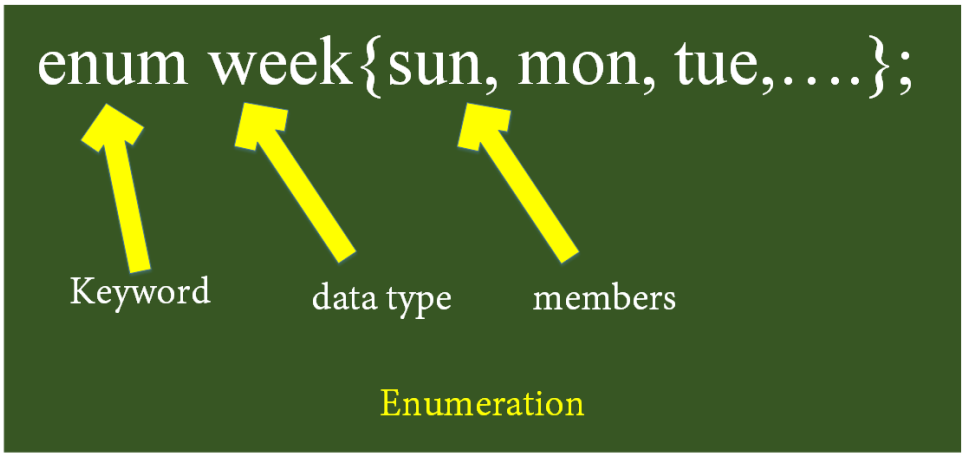
```
#include <stdio.h>

int main() {
  int number = 3;

  if (number % 2 == 0) {
    printf("Even Number");
  }
  else {
    printf("Odd Number");
  }

  return 0;
}
```

Operador ternário

```
#include <stdio.h>

int main() {
  int number = 3;

  (number % 2 == 0) ?
        printf("Even Number") :
        printf("Odd Number");

  return 0;
}
```

Enumerações

- Tipo de dados para definer um conjunto constantes
- Apresenta uma numeração para além do descritivo

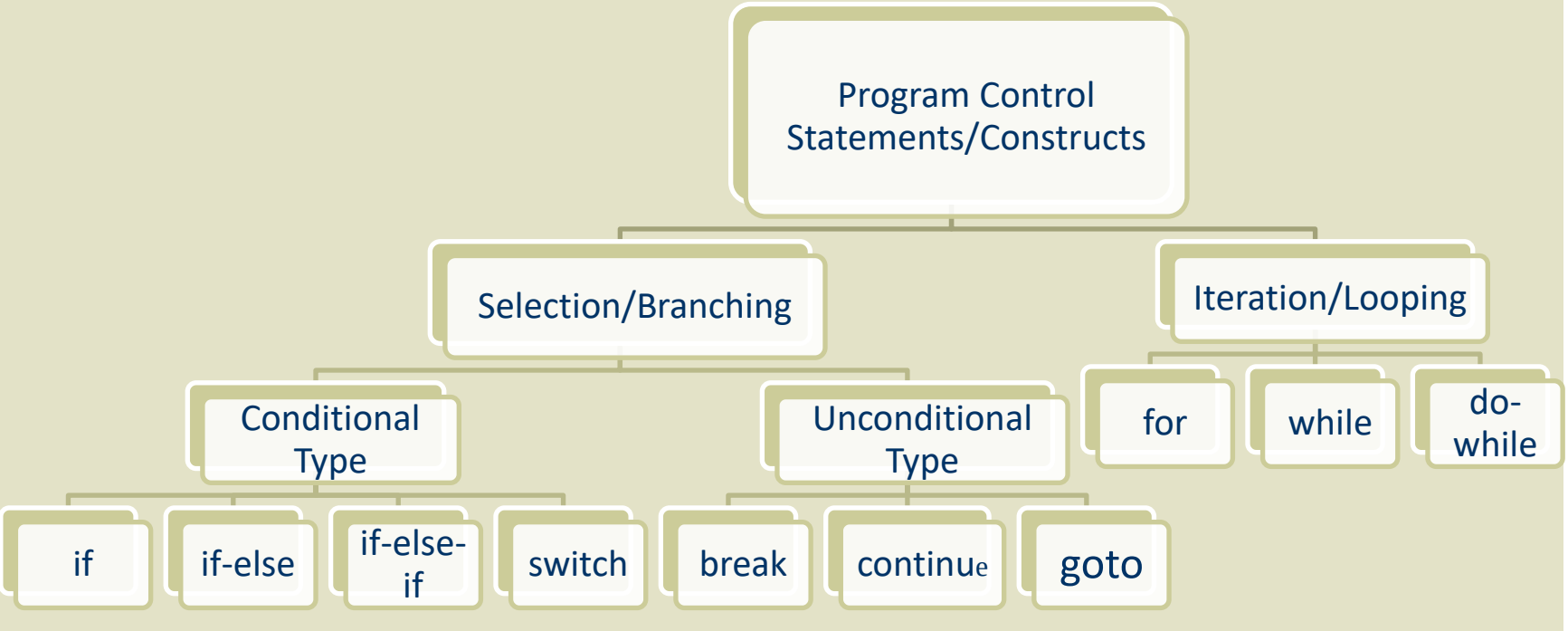


Enumerações

```c
#include <stdio.h>

// declaration on enum
enum textEditor {
    BOLD = 5,
    ITALIC = 9,
    UNDERLINE
};

int main() {
    // Initializing enum variable
    enum textEditor feature = BOLD;
    printf("Selected feature is %d\n", feature);

    // Initializing enum with integer equivalent
    feature = 5;
    printf("Selected feature is %d\n", feature);

    return 0;
}
```

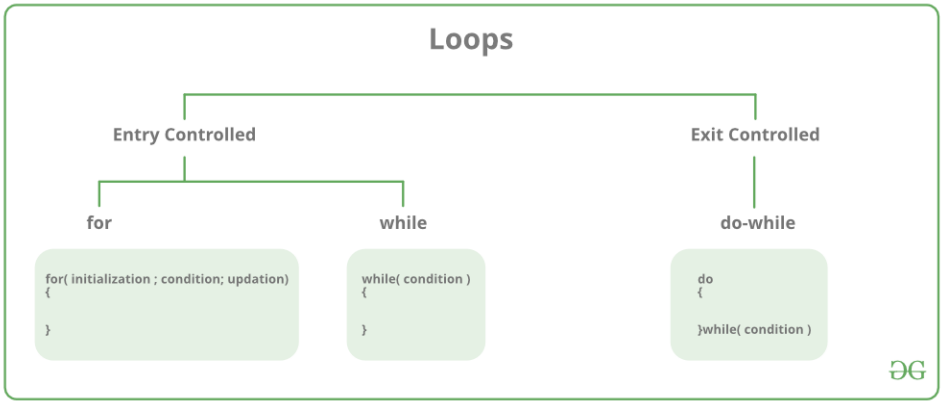
- Uma variável do tipo enumerado, pode assumir o descritivo ou o índice definido
- Só a escrita do índice pode ser efetuada
- Caso não seja definido qualquer índice é assumido o “0” no primeiro elemento
- Caso algum elemento não tenha índice, assume o índice anterior + 1

Estruturas de iterativas

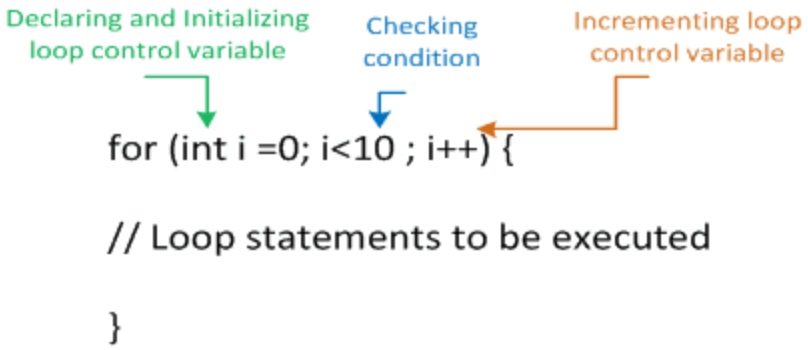


Estruturas iterativas

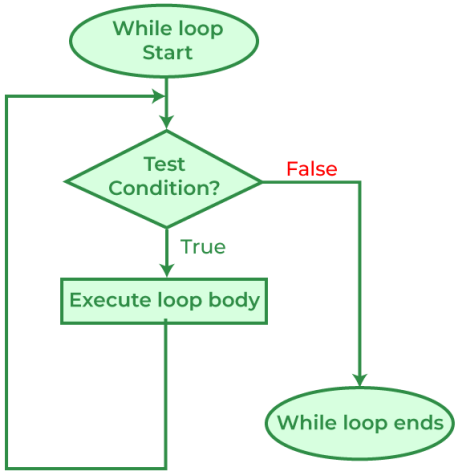
- Os *loops* são utilizados para a repetir o mesmo bloco de código até que uma condição se verifique



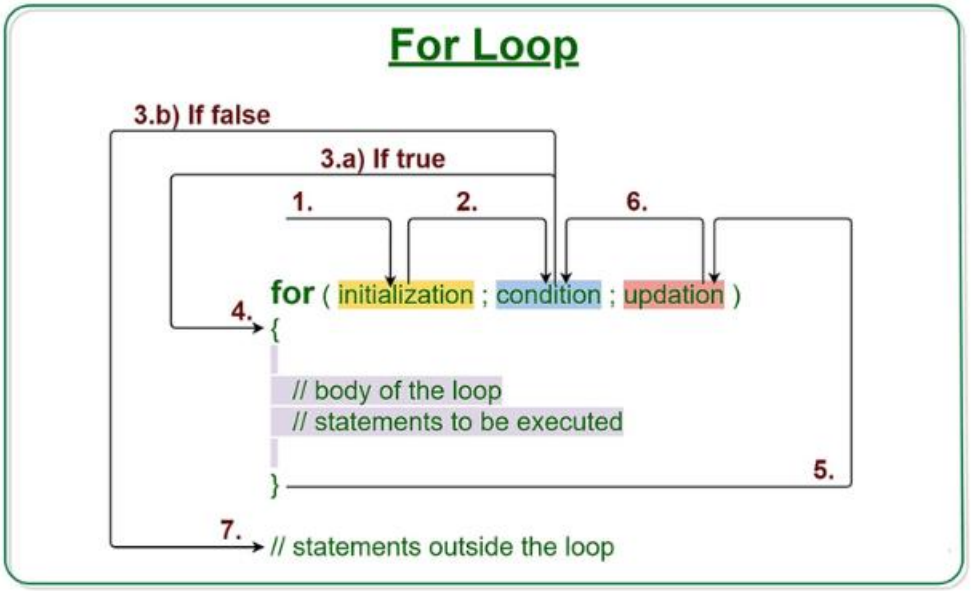
Estruturas iterativas - *for*

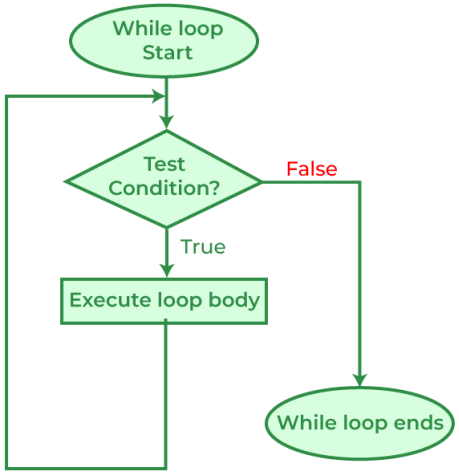



```c
#include <stdio.h>

int main () {

   int a;

   /* for loop execution */
   for( a = 10; a < 20; a = a + 1 ){
      printf("value of a: %d\n", a);
   }

   return 0;
}
```

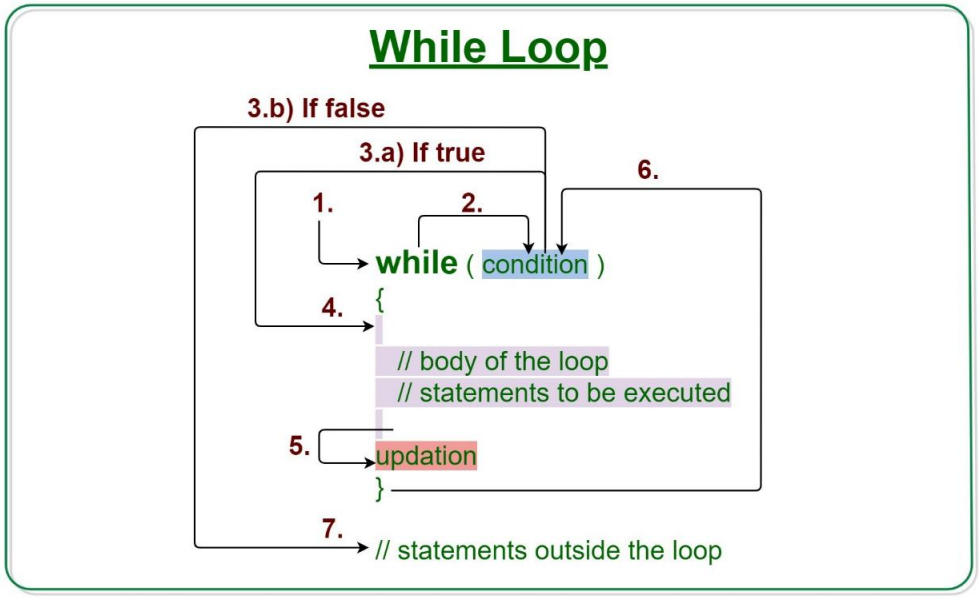


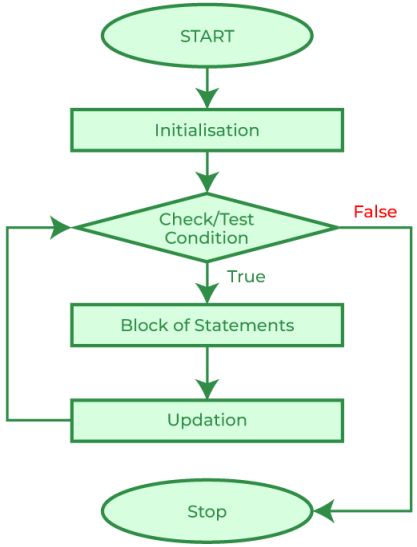
Estruturas iterativas - *for*



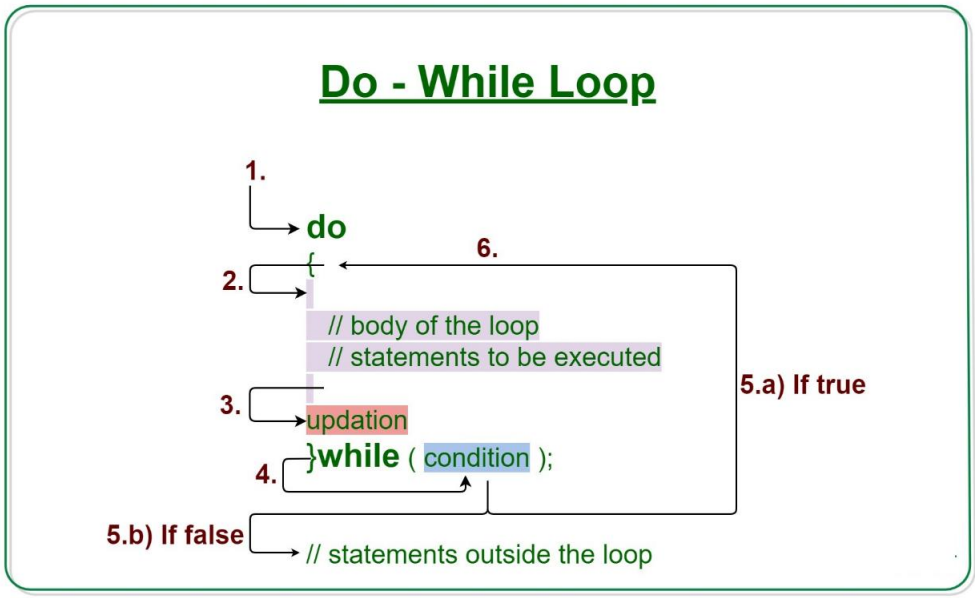


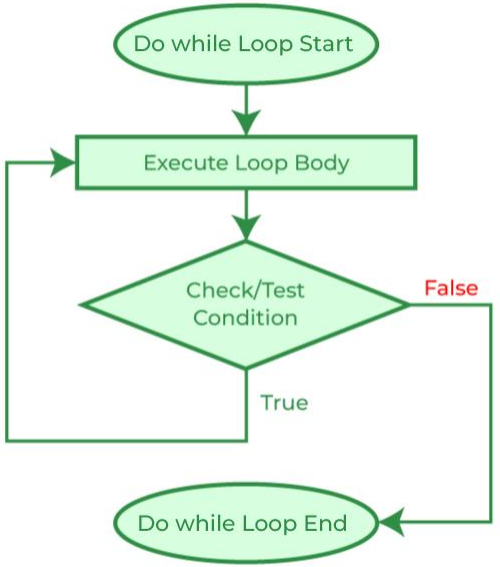
Estruturas iterativas – *while*





Estruturas iterativas – *do..while*






```c
#include <stdio.h>

int main()
{

    // loop variable declaration and initialization
    int i = 0;
    // do while loop
    do {
        printf("do While\n");
        i++;
    } while (i < 3);

    return 0;
}
```

Estruturas iterativas

```c
#include <stdio.h>

int main()
{
    int i ;

    for(i=0;i<3;i++){
        printf("For\n");
    } ;

    return 0;
}
```

```c
#include <stdio.h>

int main()
{

    // loop variable declaration and initialization
    int i = 0;
    // do while loop
    while (i < 3) {
        printf("While\n");
        i++;
    } ;

    return 0;
}
```

```c
#include <stdio.h>

int main()
{

    // loop variable declaration and initialization
    int i = 0;
    // do while loop
    do {
        printf("do While\n");
        i++;
    } while (i < 3);

    return 0;
}
```

Estruturas iterativas

```c
#include <stdio.h>
int main() {
   int i;
   for (i = 1; i <= 10; i++) {
      if (i == 5) {
        break;  // Exit the loop when i equals 5
      }
      printf("%d ", i);
   }

   return 0;
}
```

```c
#include <stdio.h>
int main() {
  int i;
  for (i = 1; i <= 10; i++) {
    if (i == 5) {
    // Skip the remaining statements in this iteration
     continue;
    }
    printf("%d ", i);
  }
 return 0;
}
```

```c
#include <stdio.h>
int main() {
  int i;
  for (i = 1; i <= 10; i++) {
    if (i == 5) {
    // Transfer control to the "skip" label
      goto skip;
    }
    printf("%d ", i);
  }

 skip:
   printf("\nSkipped number 5.");

 return 0;
}
```

```c
#include <stdio.h>
int main() {
  int i;
  for (i = 1; i <= 10; i++) {
    if (i == 5) {
      i=11;
    }
    printf("%d ", i);
  }



 return 0;
}
```

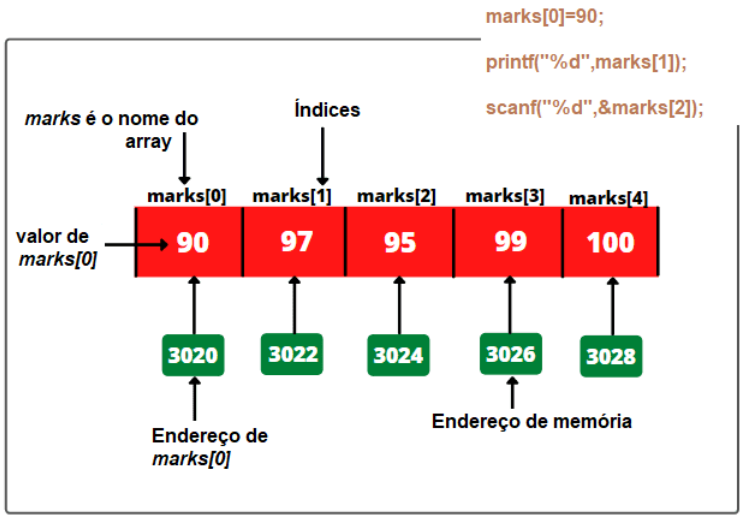
Debugging/depuração

- Permite localizar erros lógicos
- Possibilita analizar o código linha a linha
- Permite verificar o valor das variáveis
- Muitas vezes são utilizados *prints* para despistar erros

Arrays/vetores

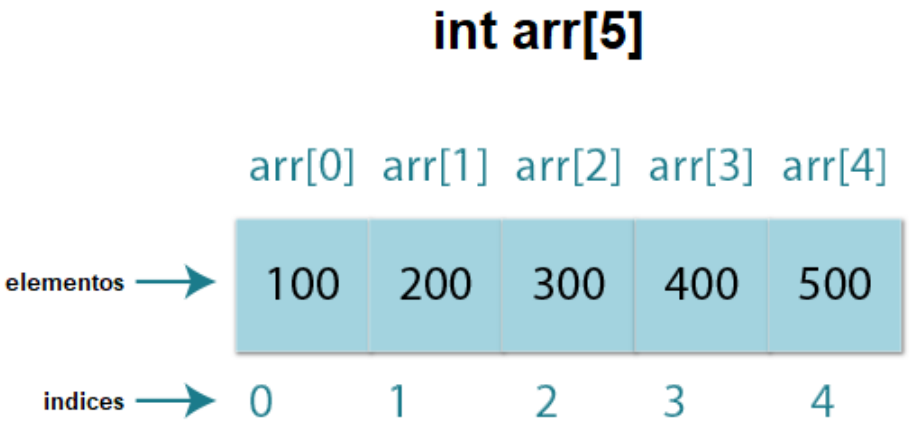
- Conjunto de elementos do mesmo tipo
- A declaração de um array informa o compilador da quantidade e tipo de elementos que compõem o *array*
- Exemplo:

  int marks[5];



Arrays/vetores

- Arrays unidimensionais




```c
#include <stdio.h>
int main() {
  int arr[5];

  arr[0]=100;
  arr[1]=200;

  printf("%d",arr[0]);

  return 0;
}
```

```c
#include <stdio.h>
int main() {
  int arr[5],i;

  arr[0]=12;
  arr[1]=121;
  arr[2]=5;
  arr[3]=3;
  arr[4]=78;

  for(i=0;i<5;i++)
    printf("%d\n",arr[i]);

  return 0;
}
```

Arrays/vetores

- Arrays multi-dimensionais

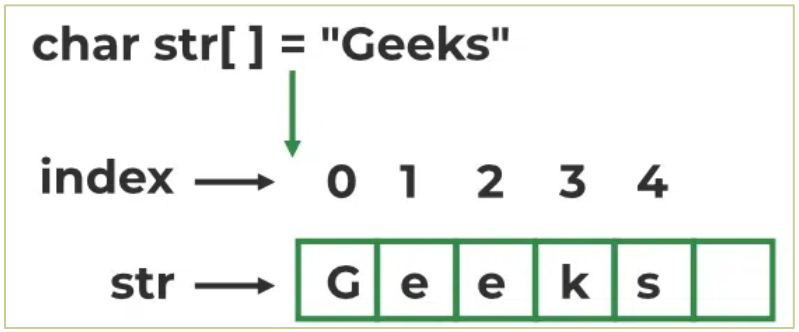
int numeros[3] [3];

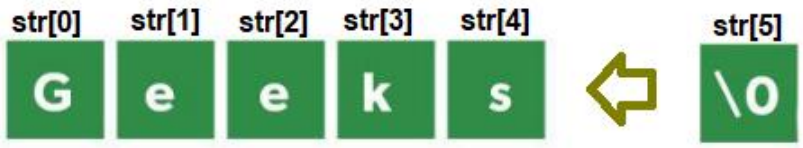
```c
#include <stdio.h>
int main() {
  int numeros[3][3];

  numeros[0][0]=10;
  numeros[0][1]=20;
  numeros[0][2]=30;

  printf("numeros[0][2]=%d",numeros[0][2]);

  return 0;
}
```

	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

Arrays/vetores

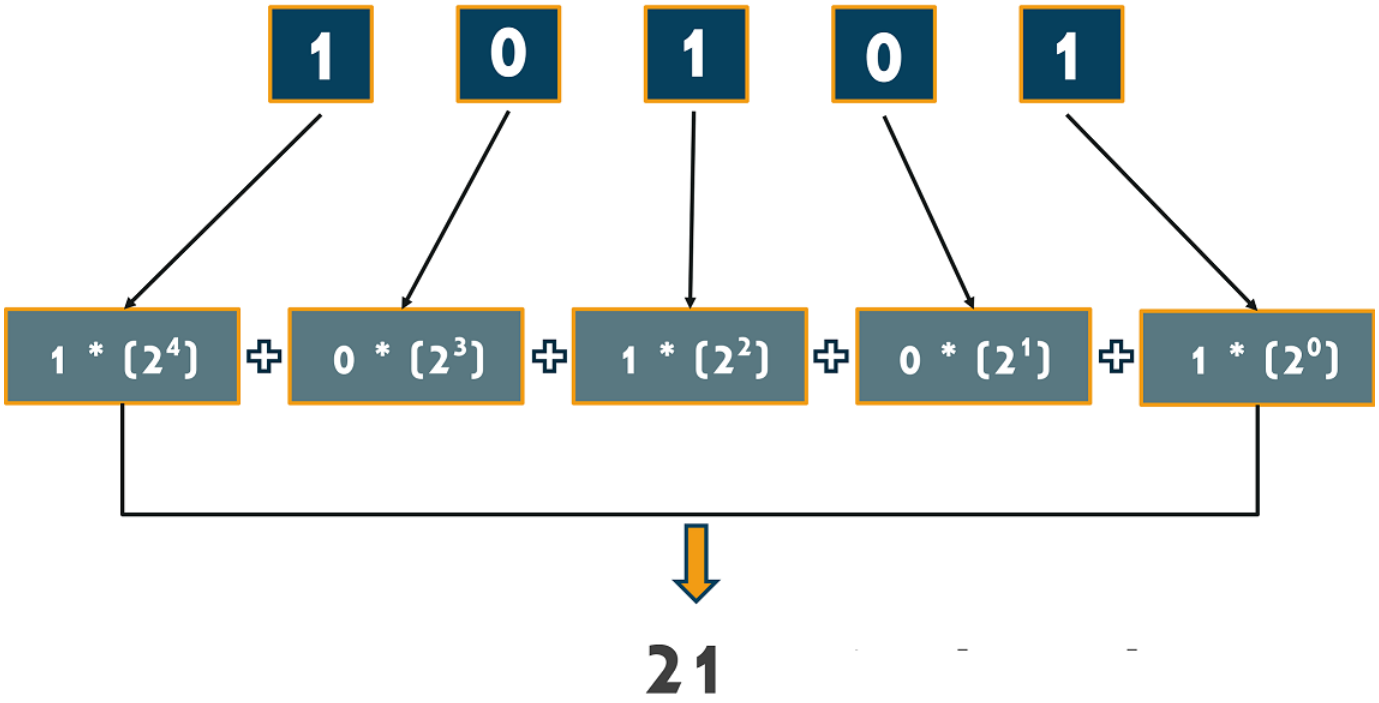
- *Strings*





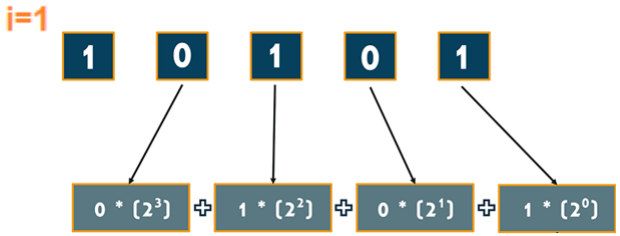

```c
#include <stdio.h>
#define MAX 15

int main(){
  char nome[MAX];

  fgets(nome, MAX, stdin);
  printf("string is: %s\n", nome);

  return 0;
}
```

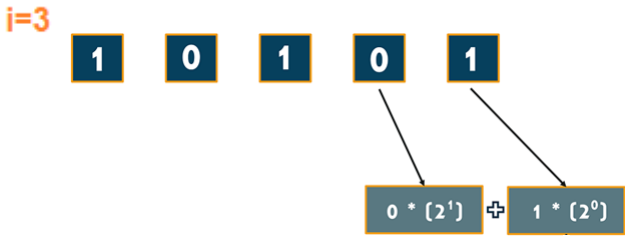
Arrays/vetores

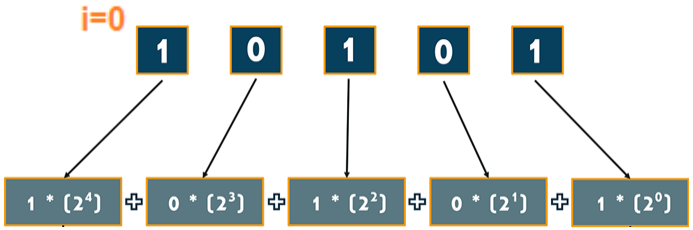


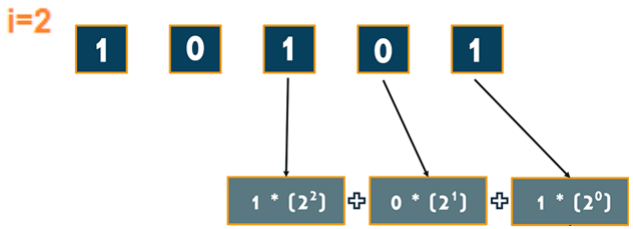
Arrays/vetores

- Conversão de binário para decimal










```
for(int i=dim;i>=0;i--)
  decimal = decimal + (arr[i]-'0')*potencia(2,dim-i);
```

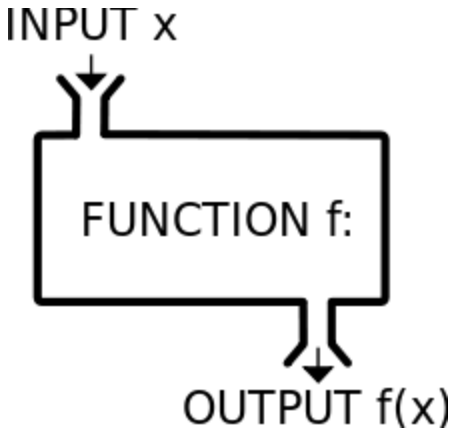
```
dim=4
i=4
  for(int i=4;i>=0;i--)
    decimal = decimal + (arr[i]-'0')*potencia(2,4-4);
```

Arrays/vetores

```
int main()
{
        int i;
        char four_planets[][8] = {"Mercury", "Venus", "Earth", "Mars"};
        for(i=0;i<4;i++)
       {
            if(four_planets[i][0] == 'M')
           {
                printf("%s Begins with M\n", four_planets[i]);
           }
        }
        return 0;

}
```

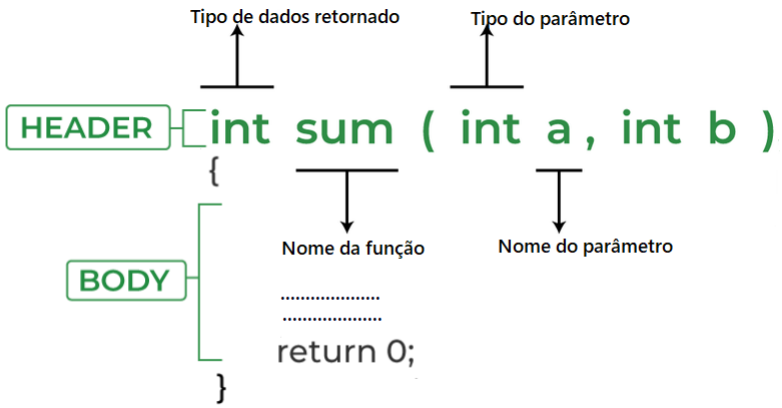
Funções/Procedimentos

- Conjunto de instruções que permite executar uma tarefa
- Bloco de código que pode ser invocado varias vezes
- Permite modular o programa

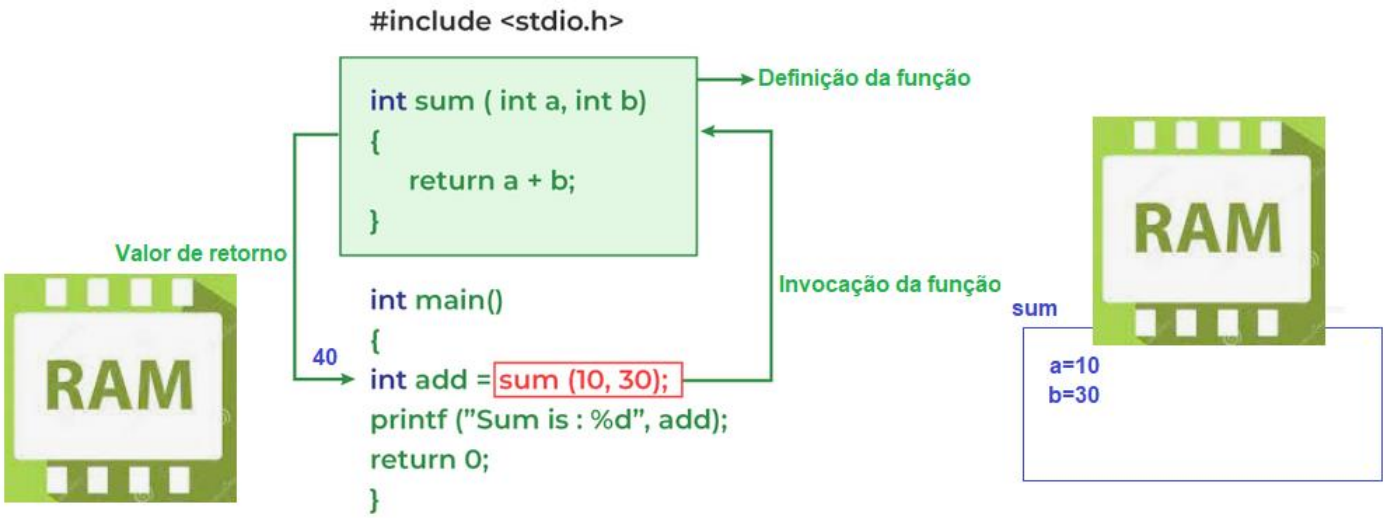


Funções

```
#include <stdio.h>

int sum( int a, int b )
{
   return a + b;
}

int main()
{

   int add = sum( 10, 30 );

   printf("Sum is: %d", add);

   return 0;
}
```

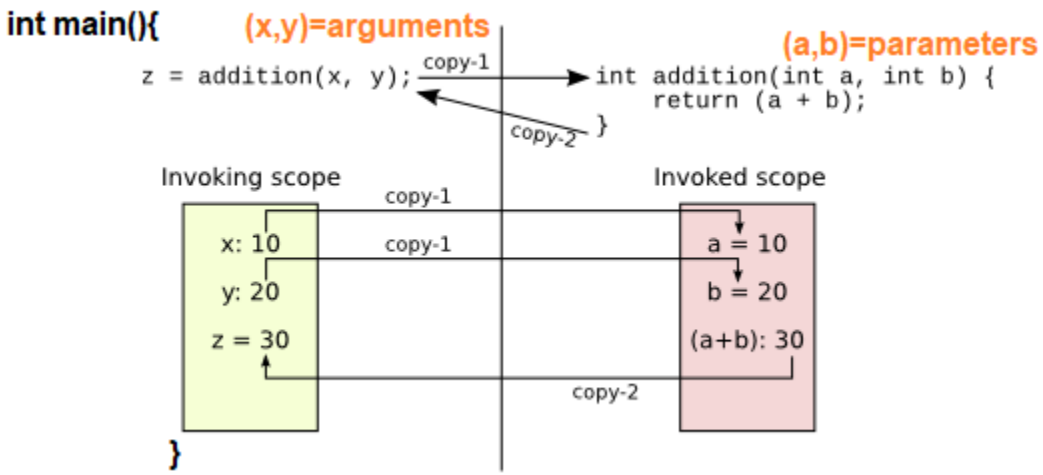


Funções


#include <stdio.h>
int sum (int a, int b)
{
return a + b;
}
Definição da função
Valor de retorno
Invocação da função
int main()
{
40
int add = sum (10, 30);
printf ("Sum is : %d", add);
return 0;
}
RAM
RAM
sum
a=10
b=30

Funções

- Passagem de parâmetros por valor
  - Consiste numa cópia dos argumentos para os parâmetros da funções



Escopo das variáveis

- Bloco de código onde a variável é válida
- As variávies são válidas no bloco onde são definidas
- As variáveis e parâmetros das funções são chamadas locais


```c
#include <stdio.h>

/* global variable declaration */
int a = 20;

int main () {

  /* local variable declaration in main function */
  int a = 10;
  int b = 20;
  int c = 0;

  printf ("value of a in main() = %d\n",  a);
  c = sum( a, b);
  printf ("value of c in main() = %d\n",  c);

  return 0;
}

/* function to add two integers */
int sum(int a, int b) {

   printf ("value of a in sum() = %d\n",  a);
   printf ("value of b in sum() = %d\n",  b);

   return a + b;
}
```

sum

a

b

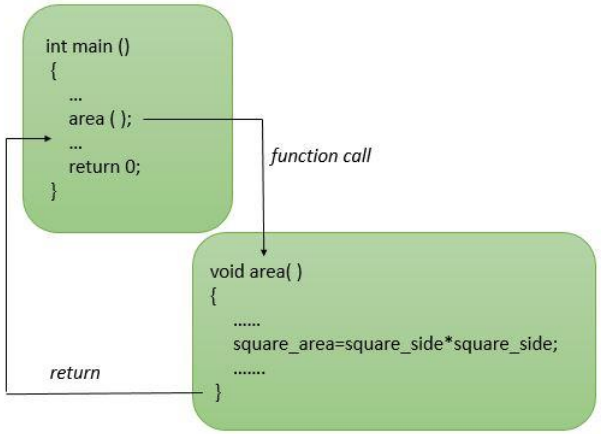
Procedimentos

- Procedimentos ou funções *void*
- Não retornam valores
- O *return* pode ser usado para terminar o procedimento

```
void  function_name() {
        block of instructions;
}
```

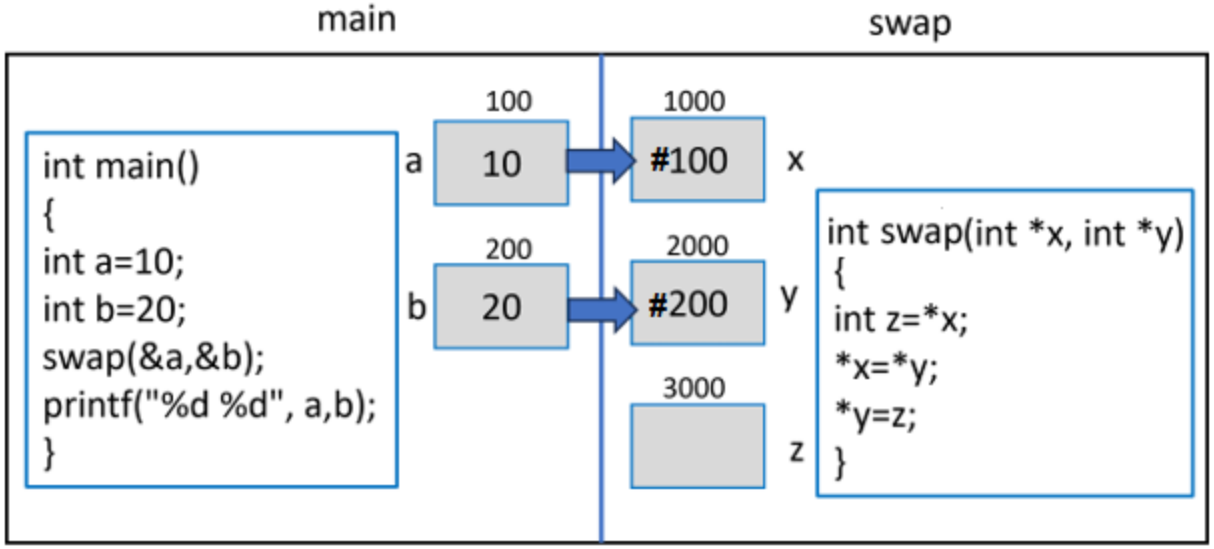
Procedimentos

```c
#include <stdio.h>

void p_dobro(int a){
  printf("Dobro de ""a""=%d\n",a*2);
}


int main(){
  p_dobro(2);
  p_dobro(3);

  return 0;
}
```



Funções/Procedimentos

- Passagem de parâmetros por referência
  - Consiste numa técnica de passagem de valores às funções
  - Esta passagem de valores é gerida por ponteiros



Tipo de dados *boolean*

- A especificação C ANSI original não possui um tipo booleano
- A revisão de de C99 acrescenta um tipo booleano
- Apenas é possível a atribuição de *true* e *false*

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

int main() {
  bool x = false;
  if (x == true) {
    printf("Valor de X: true");
  } else {
    printf("Valor de X: false");
  }

  return 0;
}
```

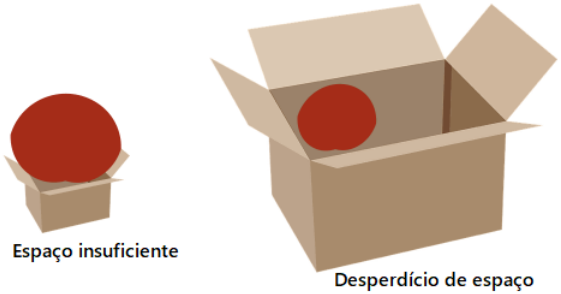
Tipo de dados *boolean*

- Tipo de boleano através de enumerações ou macros

```c
#include <stdio.h>
typedef enum {
  false,
  true
} bool_enum;


int main() {
  bool_enum x = false;
  if (x == true) {
    printf("Valor de X: true");
  } else {
    printf("Valor de X: false");
  }

  return 0;
}
```

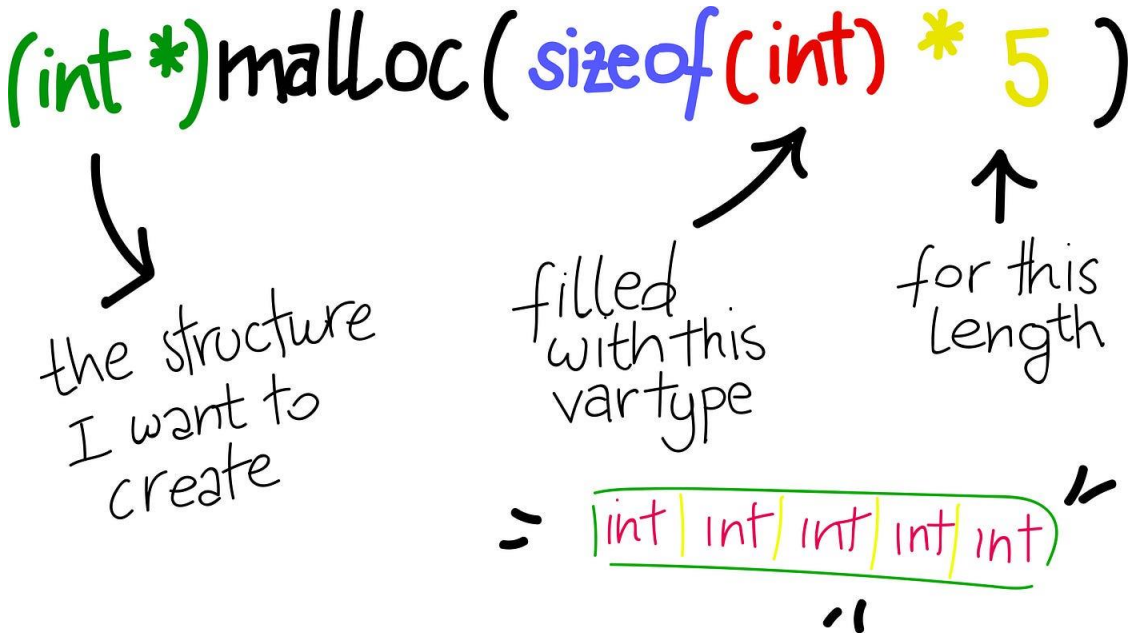
```c
#include <stdio.h>

#define true 1
#define false 0

int main() {
  int isTrue = true;
  int isFalse = false;

  printf("isTrue: %d\n", isTrue);
  printf("isFalse: %d\n", isFalse);

  isFalse=(isFalse>=0);

  printf("isFalse: %d\n", isFalse);

  return 0;
}
```

Alocação de memória dinâmica



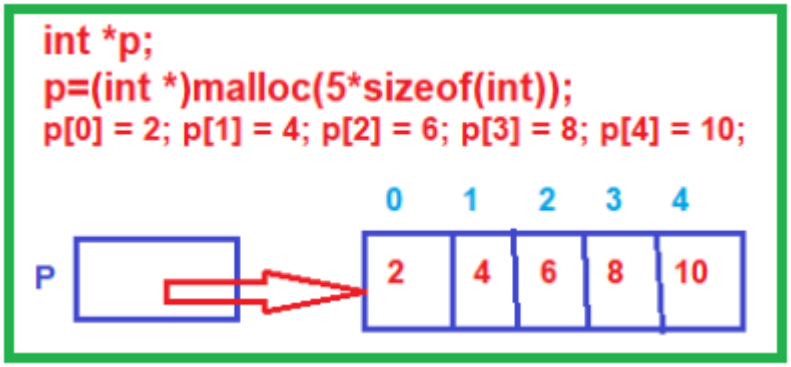
- Método de alocação que evita desperdício de memória
- A alocação da memória é feita em *run-time*
- A memória alocada pode ser alterada em *run-time*
- Se a memória não for utilizada, pode ser libertada

Alocação de memória dinâmica



Alocação de memória dinâmica

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

int main() {
    int size;

    // Get the size of the array from the user
    printf("Enter the size of the array: ");
    scanf("%d", &size);

    // Allocate memory for an integer array
    int* dynamicArray = (int*)malloc(size * sizeof(int));

    if (dynamicArray == NULL) {
        // Memory allocation failed
        printf("Memory allocation failed.\n");
        return 1;  // Exit with an error code
    }

    // Use the dynamically allocated array
    for (int i = 0; i < size; i++) {
        dynamicArray[i] = i * 10;
    }

    // Display the contents of the dynamically allocated array
    printf("Dynamically allocated array: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", dynamicArray[i]);
    }
    printf("\n");

    // Free the allocated memory
    free(dynamicArray);

    return 0;
}
```



struct(structure)

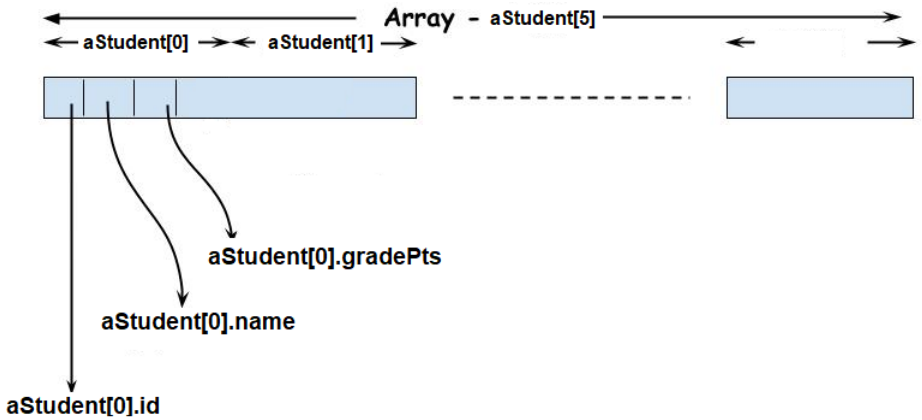
- Tipo de dados composto
- Um coleção de variáveis, de diferentes tipos
- Permitem definer tipos de dados complexos

```c
#include <stdio.h>
struct Rectangle
{
    int length;
    int breadth;
};

int main()
{
    struct Rectangle r = { 10, 5 };
    r.length = 20;
    r.breadth = 10;
    printf ("Area of Rectangle: %d", r.length * r.breadth);
    return 0;
}
```

struct(structure)

```
// Global Type Declarations
struct STUDENT
   {
    char id[10];
    char name[26];
    int  gradePts;
   } ;

// Local Declarations
struct STUDENT aStudent;
```

```
// Global Type Declarations
typedef struct
   {
    char id[10];
    char name[26];
    int  gradePts;
   } STUDENT;

// Local Declarations
STUDENT aStudent;
```

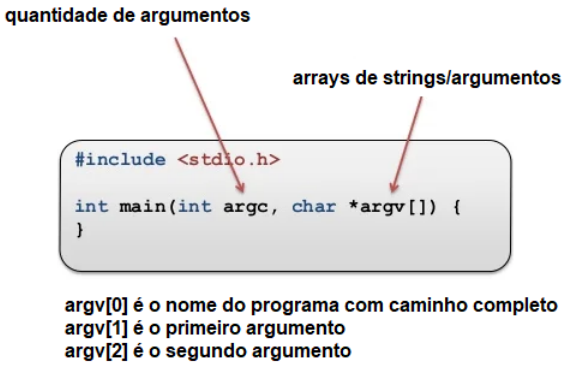


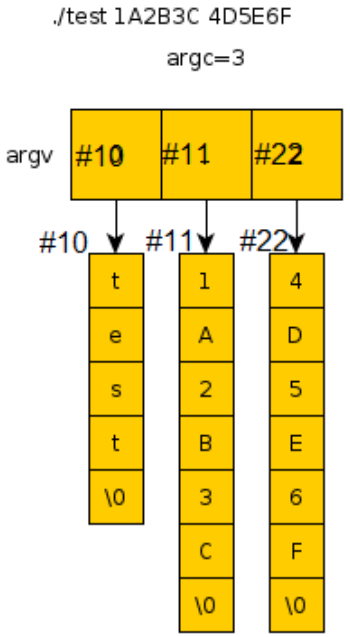
struct(structure)

```
struct Box { int width, length, height; };

int main()
{
  struct Box b, c;
  b.width = 5; b.length=1; b.height = 2;
  c = b;
  if (c == b)   /* Erro na compilação! */
    printf("c = b\n");
  else
    printf("c 1= b \n");
} t
```

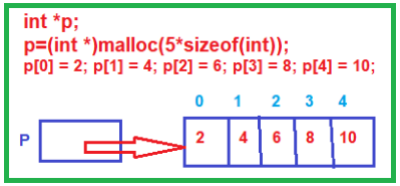
```
struct Box { int width, height , length; };

int IsEqual(struct Box b, struct Box c)
{
  if (b.width==c.width &&
     b.length==c.length &&
     b.height==c.height)
              return 1;
  else
              return 0;
}
```

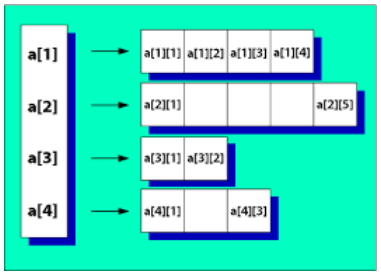
Argumentos na linha de comando





Quando conhecida a posição de memória da primeira posição do *array*, os restantes elementos são acedidos através da síntaxe de um *array*.

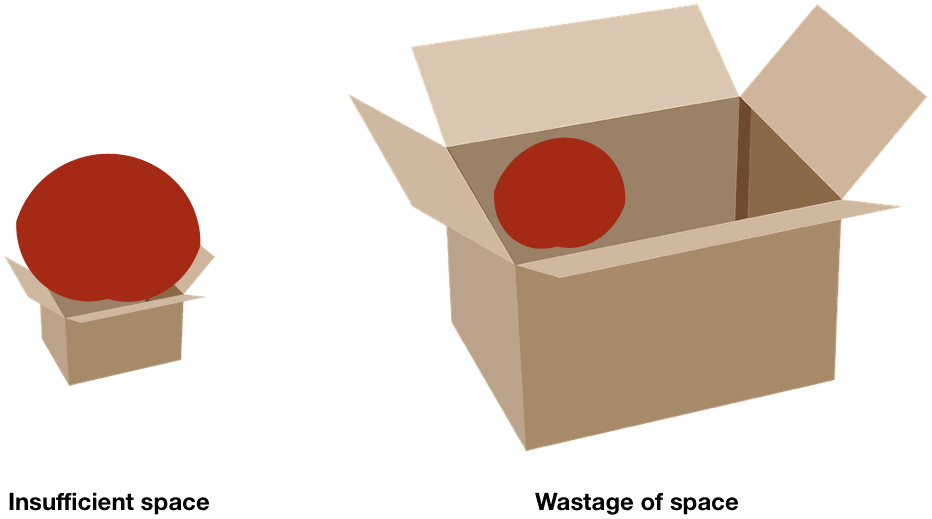




Argumentos na linha de comando

```c
#include <stdio.h>

int main(int argc, char *argv[]) {

    int i;

    printf("\nProgram name: %5", argv[0]);

    if (argc < 2) {
        printf("\n\nNo argument passed through command line!");
    } else {
        printf("\nArgument supplied: ");
        for (i = 1; i < argc; i++){
            printf("%s\t", argv[i]);
        }
    }
}
```

Alocação de memória

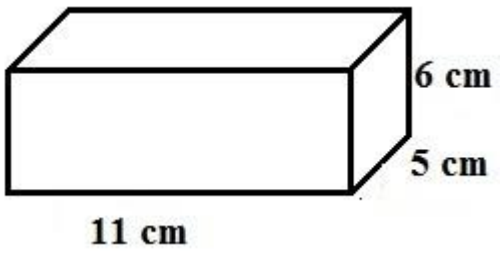


- https://www.codesdope.com/c-dynamic-memory/
- https://prepinsta.com/all-about-c-language/dynamic-memory-allocation-using-calloc/

Exemplos de código C

volume.c

```
#include <stdio.h>

int main(void) {
   int l, w, h, v;   // dimensões e volume

   l = 11;    // comprimento
   w = 5;     // largura
   h = 6;     // altura
   v = l*w*h; // cálculo do volume

   printf("LxWxH: %d*%d*%d (cm)\n", l,w,h);
   printf("Volume: %d (cm^3)\n", v);
   return 0;
}
```



Introdução do *debugging*

- Syntax Errors
- Semantic Errors
- Runtime Errors

Introdução do *debugging*

- Syntax Errors
- Semantic Errors
- Runtime Errors

Common Errors

```
#include <stdio.h>
int main(void)
(
    int n  int n2  int n3;
    /* this program has several errors
    n = 5;
    n2 = n * n;
    n3 = n2 * n2;
    printf("n =  %d  n squared = %d  n cubed = %d\n"  n  n2  n3);
    return 0;
)
```

Data and C

- Programs work with data.
- A common C program works like this
  - You feed data to your program.
  - Your program does something with the data.
  - Your program gives the result back to you.

Example Reading Input from Keyboard

```
int main()
{
   float weight;
   scanf("%f"  &weight)
   printf("george's weight is %f.\n" weight);
   return 0;
}
```

Float and Int

- Bits Bytes and Words.
- The integer

+/-	0	0	0	0	1	1	1

- The Float

+/-	.314159	1

Type Char

- The char type is used for storing characters such as letters and punctuation marks.
- Char type actually stored as integer (length 1 byte)
- Example

```
char broiled;   //declare a char variable.
broiled = 'T';  //correct
broiled = T;    //error
broiled = "T";  //error
```

Character strings

- An example of a string

  “I am a string.”
- A character string is a series of one or more characters.
- Strings are enclosed by double quotation marks.

Character strings(2)

- C has no special string type
- A string is an array of chars
- Characters in a string are stored in adjacent memory cells
- Standard C string functions depend on a null terminated string

h	i		t	h	e	r	e	\0	

Character strings (3)

- String declaration

```
char name[5];
```

- Notice the difference

```
char ch;
char name[5];
```

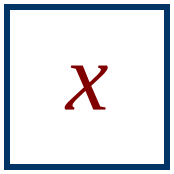
- Every char of name can be accessed as name[i]
- Arrays are indexed from 0 so the first character in a string is `string[0]`

Sample program

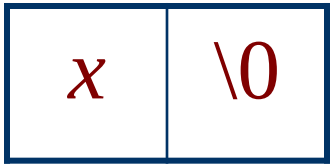
```
int main()
{
    char name[40];
    printf("what is your name?");
    scanf("%s"  name);
    printf("hello  %s.\n"  name);
    return 0;
}
```

Strings versus characters

- Character ‘x’



- String “x”



Common String functions

- É<string.h>
- *strlen* // returns the length of the string
- *strcpy* // string copy
- *strcmp* // string compare
- *strcat* // append one string to another
- *sprintf* // same as *printf* but prints to a string
- *sscanf* // same as *scanf* but reads from a string

Question

- What does strlen () return if applies to the following string? Why?

  “hello everybody\0 my name is dr. Evil.”

Function Structure

Return type Function name (arguments)

{

    Function body

}

More on functions

- A function is a self-contained module of code that can accomplish some task.
- Example:

```
int myfunction(int a)
{
    return (a + 1);
}
```

Operators

- Arithmetic

addition	**+**
subtraction	**-**
multiplication	*****
division	**/**
modulus	**%**

integer addition is not the same as floating point, be careful with types

- Assignment **=**

```
eg. Number = 23;
```

- Augmented assignment

+= -= *= /= %= &= |= ^= <<= >>=

eg.

```
Number += 5;
```

is equivalent to

```
Number = Number + 5;
```

Operators

- bitwise logic
  - NOT `~`
  - AND `&`
  - OR `|`
  - XOR `^`
- bitwise shifts
  - shift left `<<`
  - shift right `>>`
- boolean logic
  - Not `!`
  - And `&&`
  - Or `||`

Example:

```
int num1 = 1, num2 = 2, result;

result = num1 && num2; // result = 1
result = num1 & num2;  // result = 3
```

Note: there is no boolean type, non-zero is considered logically true

Operators

- equality testing
  Equal to **==**
  Not equal to **!=**
- order relations **< <= > >=**
- conditional evaluation `(expr) ? … result1 : result2;`
  example:

```
int num1 = 5;
result = num1==5 ? 1 : 2 // result = 1
```

  is equivalent to

```
if (num1 == 5) result = 1 else result = 2;
```

  note the difference between

```
num1 = 5;     // assignment
num1 == 5;    // logical test
```

- increment and decrement **++ --**
  order can be important: ++i and i++ are both valid

- object size **`sizeof ()`** – NOT the same as **`strlen ()`**

Loops

- for loop

```
for (i = 0; i < max; i++)
    {… body… }
```

- while loop

```
while (expression)
    {… body …}
```

- *do loop*

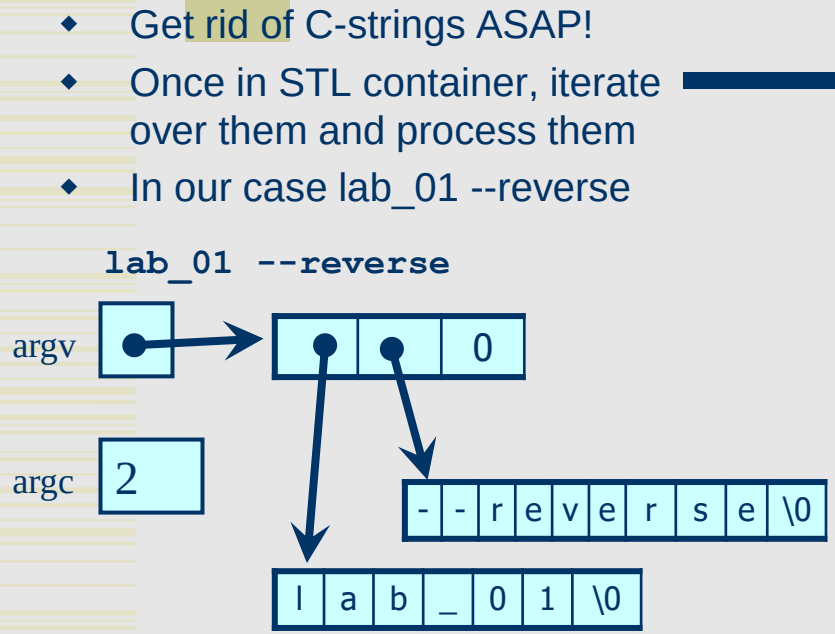
```
do
{… body… }
while (expression);
```

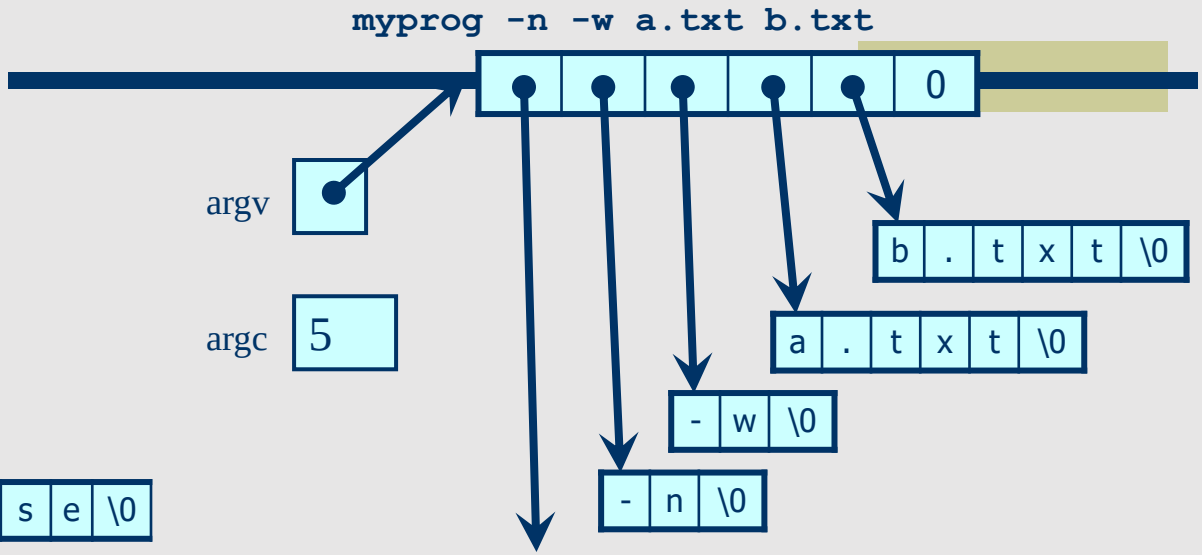
Example

```
int main()
{
    int i = 0;
    while (i < 3)
    {
        printf("%d "  i);
        i = i + 1;
    }
    return 0;
}
```

Coding: Processing cmd args

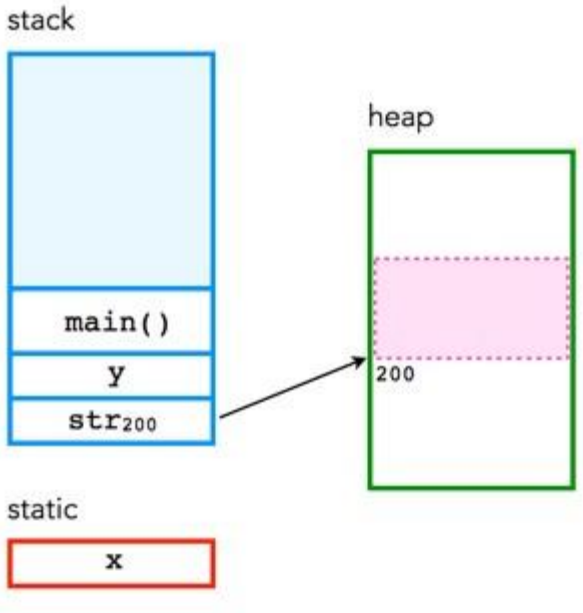
- Get rid of C-strings ASAP!
- Once in STL container, iterate over them and process them
- In our case lab_01 --reverse





Alocação de memória

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

int x;

int main(void)
{
    int y;
    char *str;

    y = 4;
    printf("stack memory: %d\n", y);

    str = malloc(100*sizeof(char));
    str[0] = 'm';
    printf("heap memory: %c\n", str[0]);
    free(str);
    return 0;
}
```



- – static: global variable storage, permanent for the entire run of the program.
- – stack: local variable storage (automatic, continuous memory).
- – heap: dynamic storage (large pool of memory, not allocated in contiguous order).