Fundamentos C/C++

- História do C
- Utilização do C
- IDE
- Compilação e execução
- Síntaxe C
- Primeiro programa
- Escrita de dados
- Tipos de dados
- Variáveis
- Leitura de dados
- Operadores relacionais, lógicos e aritméticos
- Constantes
- *Header Files*
- Introdução aos ponteiros
- Enumerações
- Estruturas de controlo
- Estruturas iterativas

Hugo Dias
dias.hugo@gmail.com

História do C

- Desenvolvido por Dennis M. Ritchie, na *Bell Labs*, no início dos anos 70's
- Deriva de uma linguagem chamada B
- Inicialmente implementada como linguagem do Sistema operative UNIX
- Baseada em funções, *weakly typed*, síntaxe gramática própria e *case-sensitive*
- Linguagem tanto de "alto nível" quanto de "baixo nível"

Características do C

- Apesar de muitas linguagens atuais, o C ainda é largamente utilizado como a linguagem preferencial de sistemas operativos
- C é utilizado em : *Embedded Systems*(sistemas integrados como *arduino*), a computação científica
- Características: eficiente, flexível, programação de baixo nível

Aplicação da linguagem C

```
| Select account you wish to transer from.      |
-------------------------------------------------
| 1. Checkings to Savings                       |
| 2. Savings to Checkings                       |
-------------------------------------------------

1
Enter transfer amount: $1000
Your updated Checkings balance is: $1700

----------------------------
|What would you like to do? |
----------------------------
| 1. Deposit               |
| 2. Withdraw              |
| 3. Check Balance         |
| 4. Transfer Money        |
| 5. [Exit]                |
----------------------------
3
-------------------------------------------------
|       Check Account Balance for?              |
-------------------------------------------------
| 1. Checking                                   |
| 2. Savings                                    |
-------------------------------------------------
2
Your current Savings balance is $1000
```

```
SELECT MAX(Price) AS [HighestInvoice]
FROM OrderDetails
WHERE InvoiceMonth IN ('May', 'June','July')
```

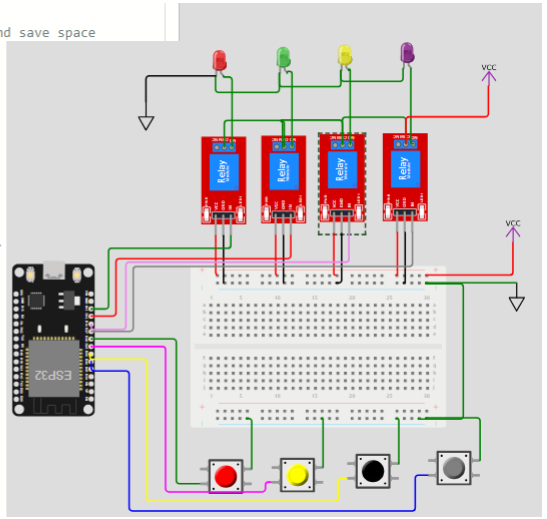
100 %

Results | Messages

	HighestInvoice
1	8999.00

Aplicação da linguagem C

```
6
7   // Comment this out to disable prints and save space
8   #define BLYNK_PRINT Serial
9
10
11  #include <WiFi.h>
12  #include <WiFiClient.h>
13  #include <BlynkSimpleEsp32.h>
14
15  char auth[] = BLYNK_AUTH_TOKEN;
16
17  // Your WiFi credentials.
18  // Set password to "" for open networks.
19  char ssid[] = "Wokwi-GUEST";
20  char pass[] = "";
21
22  BlynkTimer timer;
23
24
25  #define button1_pin 26
26  #define button2_pin 25
27  #define button3_pin 33
28  #define button4_pin 32
29
30  #define relay1_pin 13
31  #define relay2_pin 12
32  #define relay3_pin 14
```




```
#include <linux/module.h>
#include <linux/moduleparam.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <asm/uaccess.h>
#define alpha           "abcdefghijklmnopqrstyvwxyz"
#define BUF_SIZE        32

/*
        Do while having super-user privillages:
-------------------------------------------------------------------------
|       ' insmod mine.ko' :     insert module into the kernel.|
|       ' rmmod mine.ko'  :     remove module from the kernel.|
-------------------------------------------------------------------------
*/
// The entry will be created into the ' /proc ' directory
// the directory that will hold the new device.
static struct proc_dir_entry *ent;
static char message[BUF_SIZE];

static ssize_t mwrite(struct file *file, const char __user *ubuf, size_t count, loff_t *offset)
{
        unsigned int i;
        int rv;
        char __user *p = ubuf;

        printk(KERN_INFO "Write Handler\n");
        printk(KERN_INFO "Size: %d , Offset: %d \n", count, *offset);

        if(count > BUF_SIZE)
                return -EFAULT;

        rv = copy_from_user(message, p, count);

        printk(KERN_INFO "Byte not copied: %d\n",rv);
        printk(KERN_INFO "device have been written\n");

        return count;
}

static ssize_t mread(struct file *file, char __user *ubuf, size_t count, loff_t *ppos)
{
        char __user *ptr;

        printk(KERN_INFO "Read Handler\n");
        printk(KERN_ALERT "Count : %d\n",count);
        ptr = ubuf;

        if(count > BUF_SIZE) {
                printk(KERN_INFO "Adjusting size:\n");
                count = BUF_SIZE;
                printk(KERN_INFO "Size: %d\n",count);
                return (-1);
        }
        copy_to_user(ubuf, message, count);

        return count;
}

// File operations.
static struct file_operations fops =
{
        .owner = THIS_MODULE,
        .read  = mread,
        .write = mwrite,
```

Pré-requisitos

- Sistema operativo Linux/Windows
- Conhecimento básico sobre o Sistema de ficheiros
- Conceitos básicos de programação

IDE para C

- VSCODE
  - sudo apt-get install libc6-dev
  - Instalação do VSCODE no Mint
  - Extensões “C/C++ Extension Pack” e “Code Runner”
- Code::Blocks, Eclipse, NetBeans
- Utilização do compilador gcc apartir do terminal

Compilador C para Windows

Instalar o Visual Studio Code e MinGW

1- Baixar e instalar o *MinGW (Minimalist GNU for Windows)*

- ✓ https://sourceforge.net/projects/mingw/
- ✓ Install, Continue, Continue
- ✓ Selecionar "mingw32-gcc-g++", Mark For Installation"
  Menu "Installation", "Apply Changes" e "Apply"

2- Download e instalação do VSCode

3- Instalar as extensões para C/C++

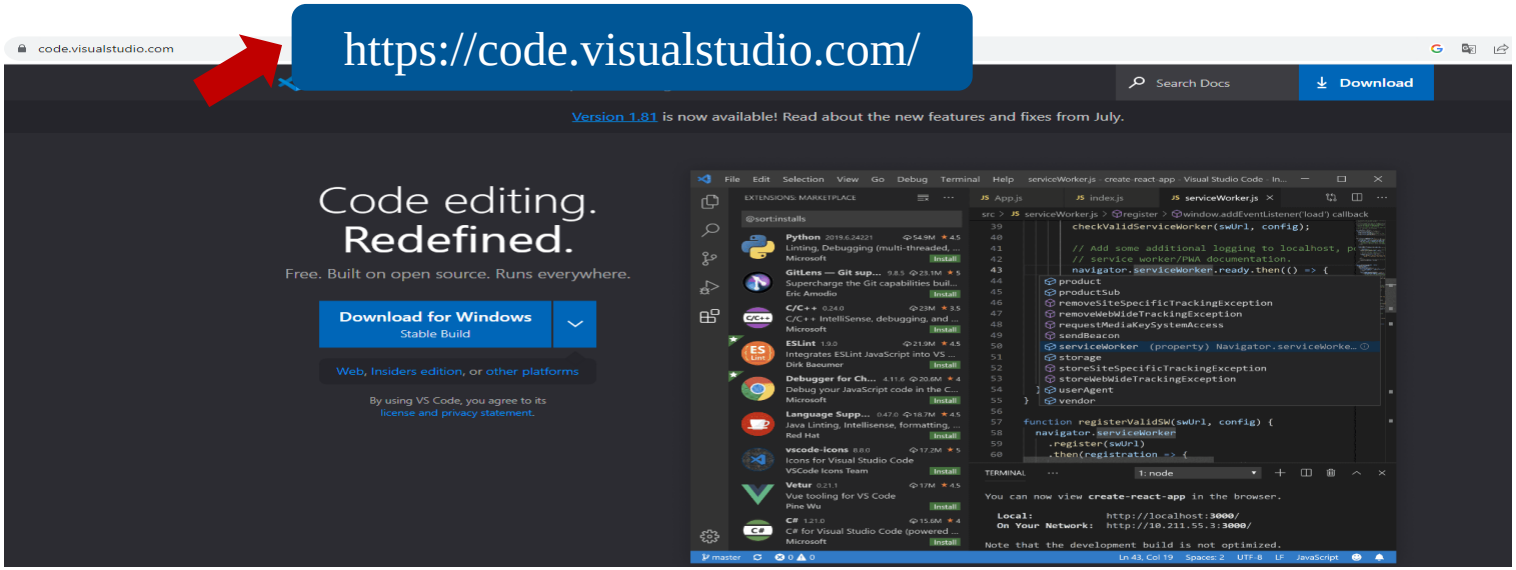
- ✓ C/C++ for Visual Studio Code
- ✓ C/C++ Extension Pack
- ✓ Code Runner - configurar a extensão para Code Runner In Terminal

Compilador C para Windows

Download e instalação do VSCode

Passo 1: Selecionar a versão a instalar

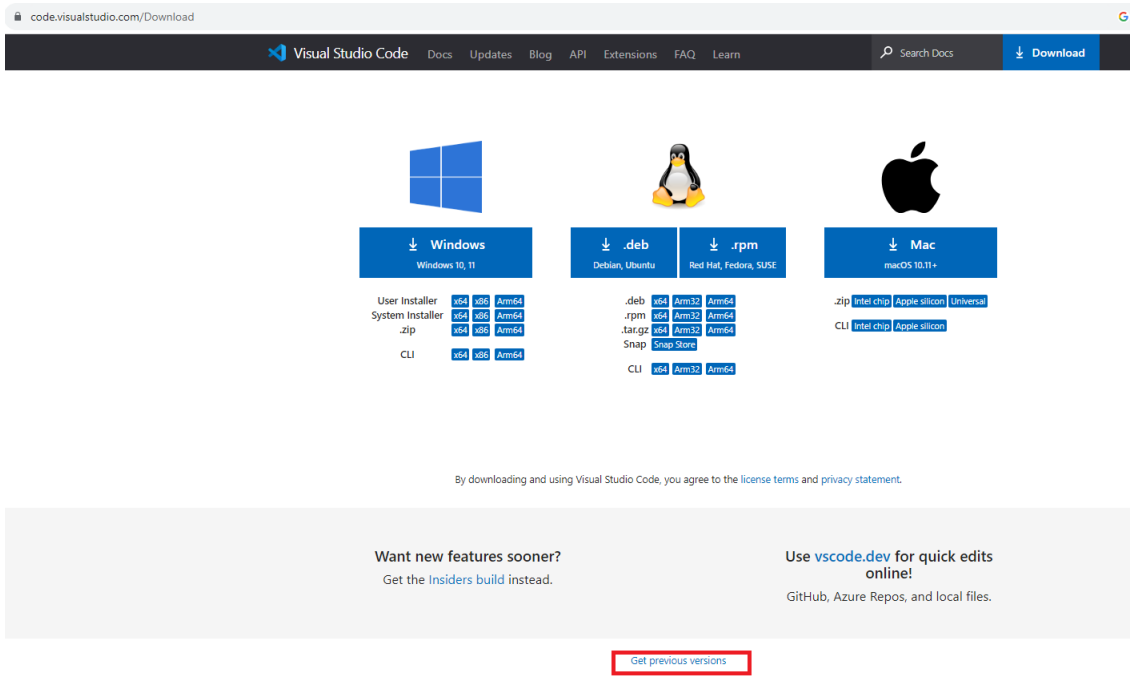
>"Download"



Compilador C para Windows

Download e instalação do VSCode

>"Get previous versions"

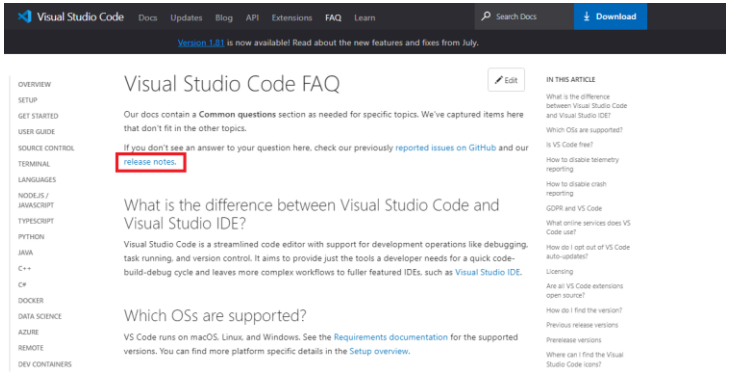


Compilador C para Windows

Download e instalação do VSCode

Compilador C para Windows

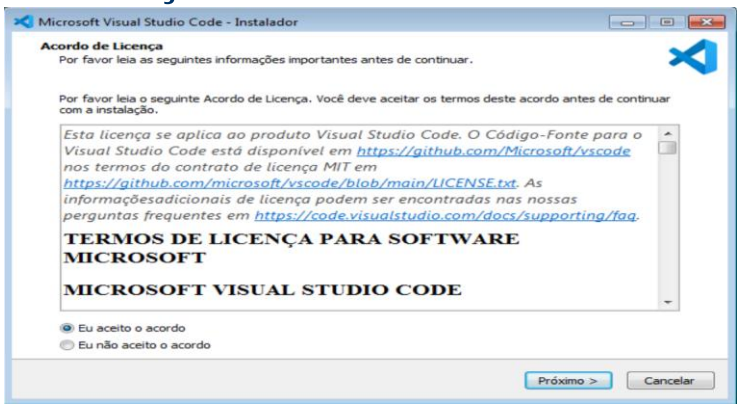
Download e instalação do VSCode

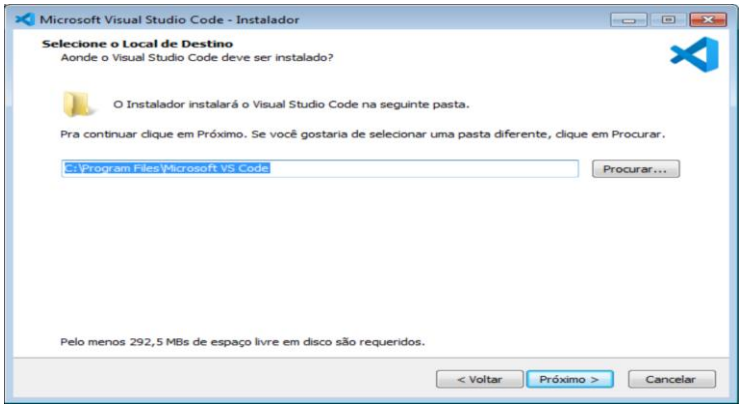


Compilador C para Windows

Download e instalação do VSCode

Passo 2: Instalação do executável

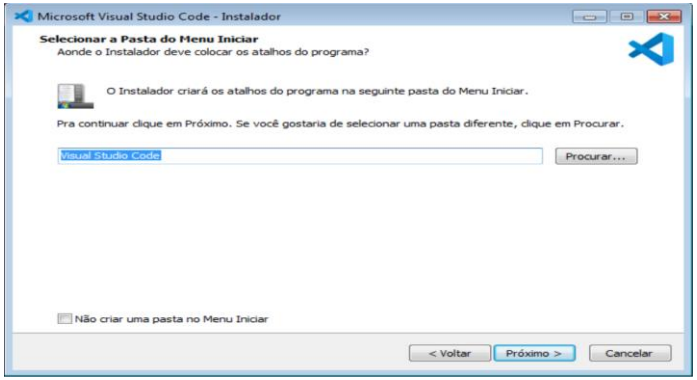


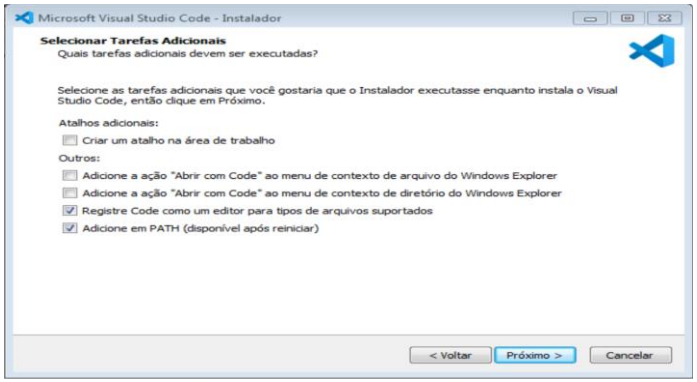


Compilador C para Windows

Download e instalação do VSCode

Passo 2: Instalação do executável

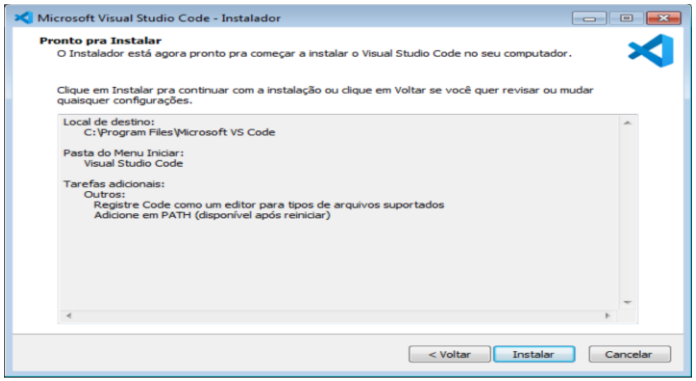


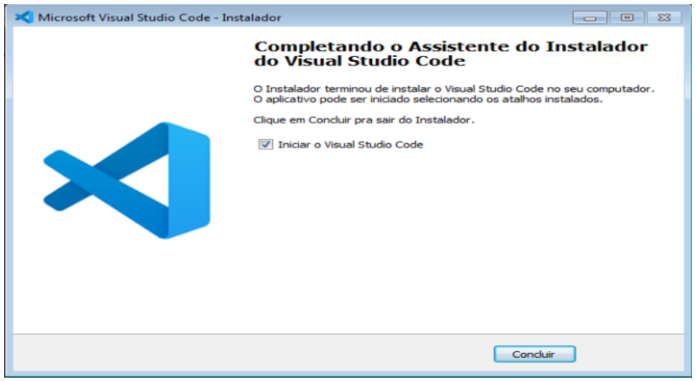


Compilador C para Windows

Download e instalação do VSCode

Passo 2: Instalação do executável

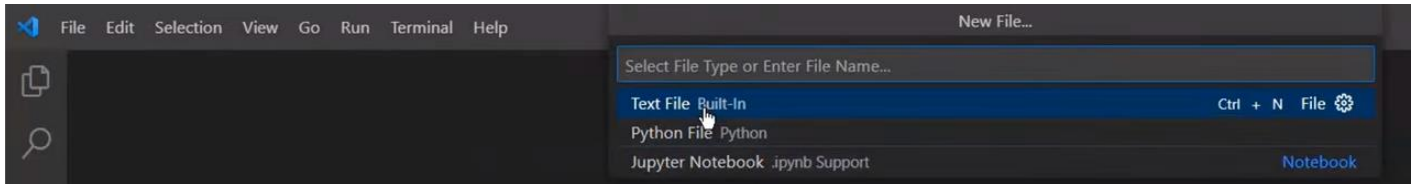




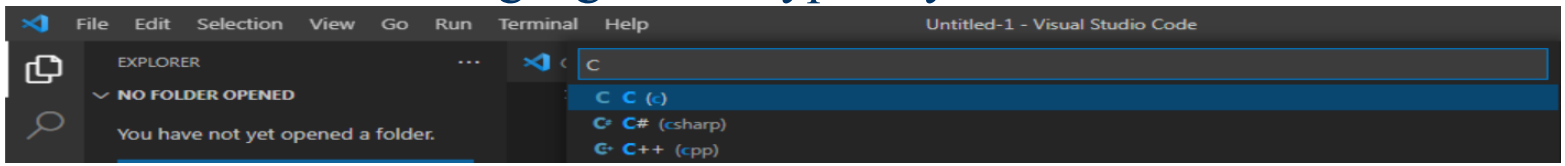
Compilador C para Windows

Download e instalação do VSCode

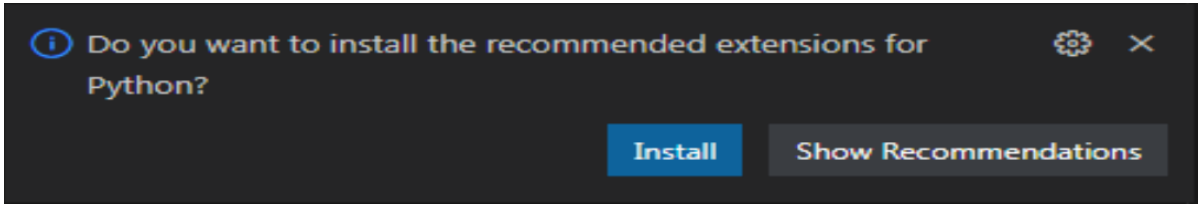
Passo 1: ”File”>”New File”



Passo 2: ”Select a Language” and type “Python”



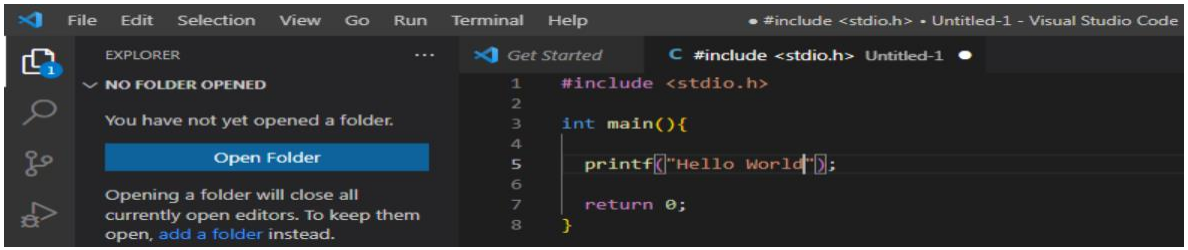
Passo 3: Caso as extensões não existam: ”*Install*”



Compilador C para Windows

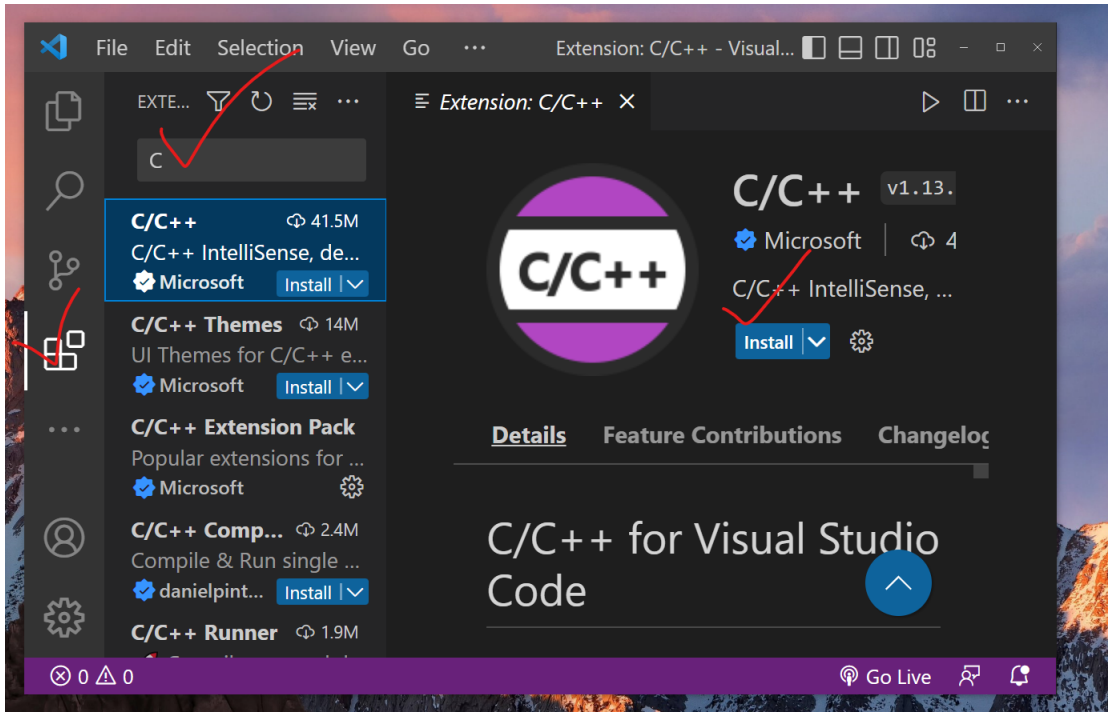
Download e instalação do VSCode

Passo 4: VSCODE pronto para o código Python



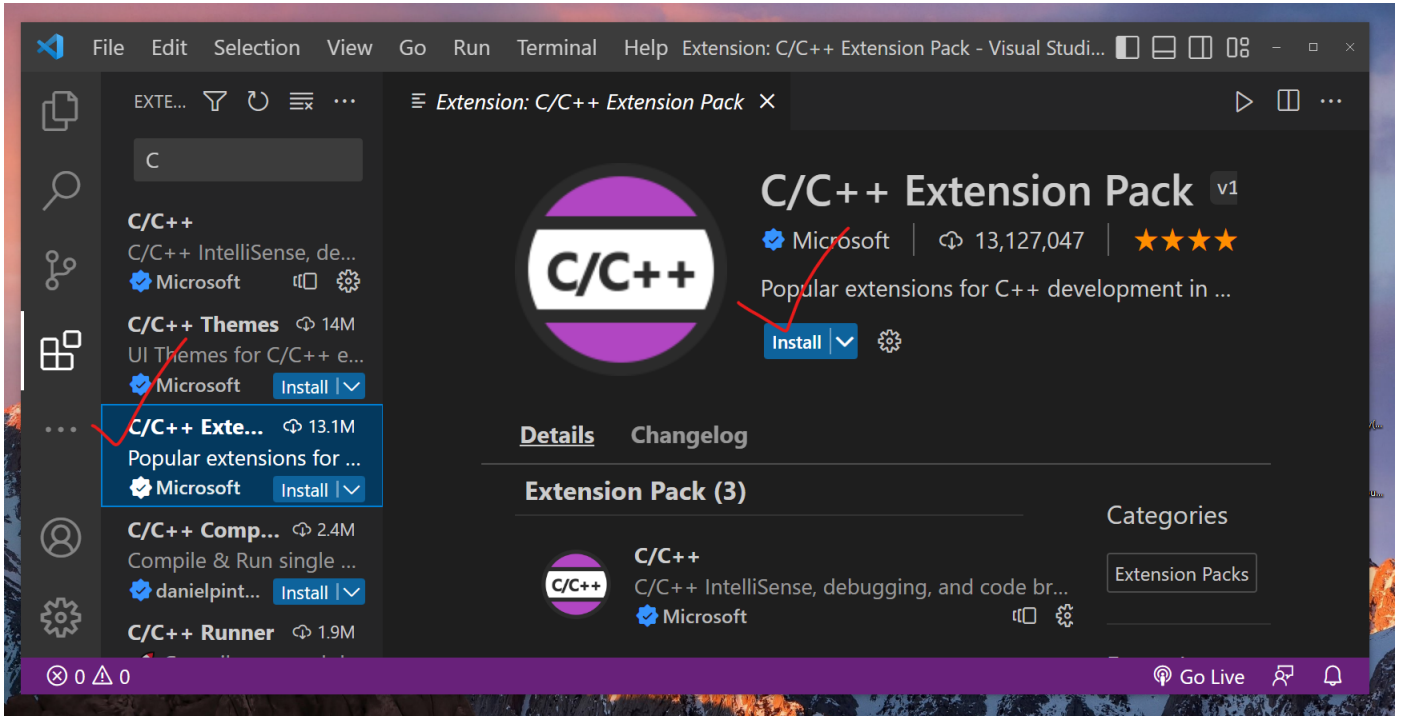
Compilador C para Windows

Instalar as extensões para C/C++



Compilador C para Windows

Instalar as extensões para C/C++

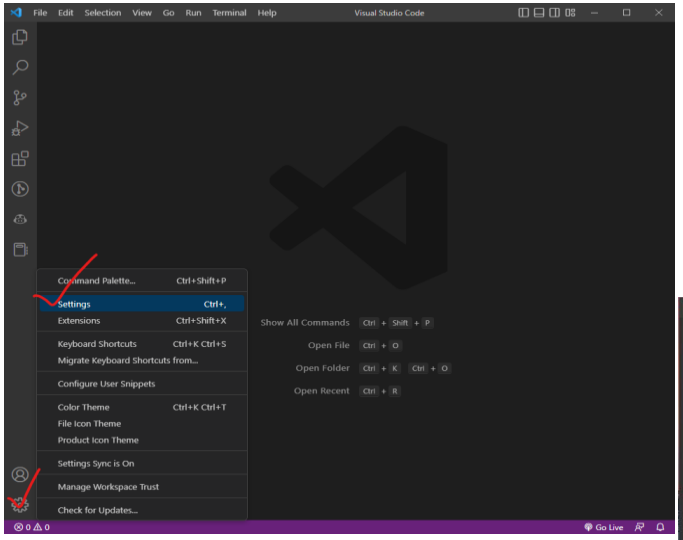


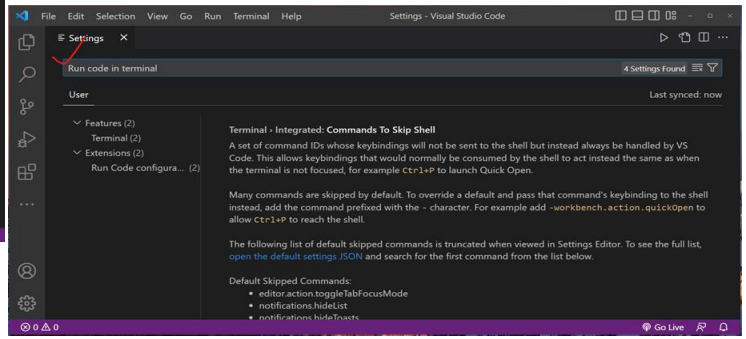
Compilador C para Windows

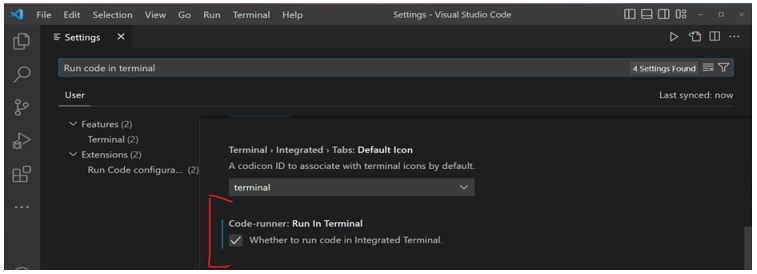
Instalar as extensões para C/C++

Compilador C para Windows

Instalar as extensões para C/C++

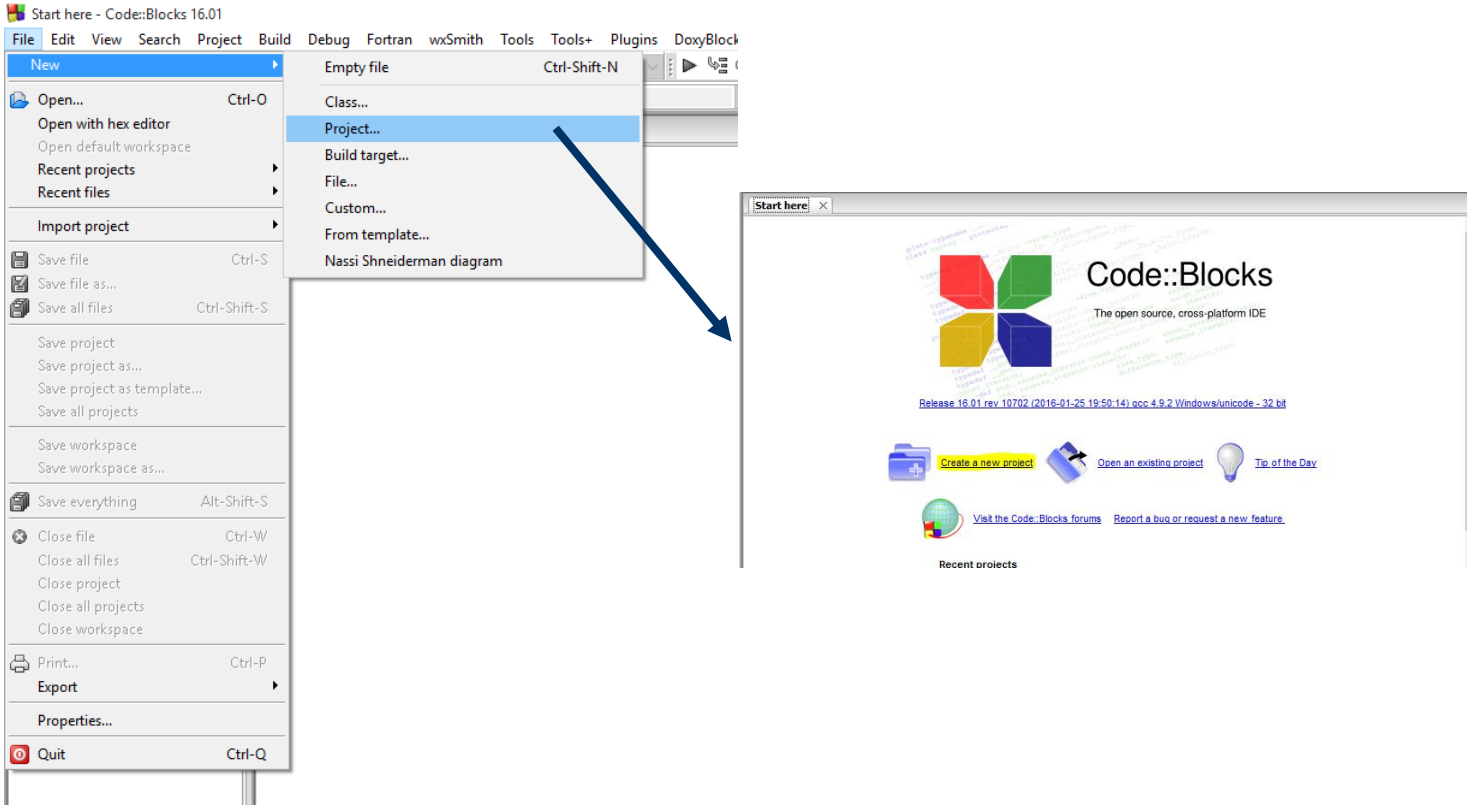






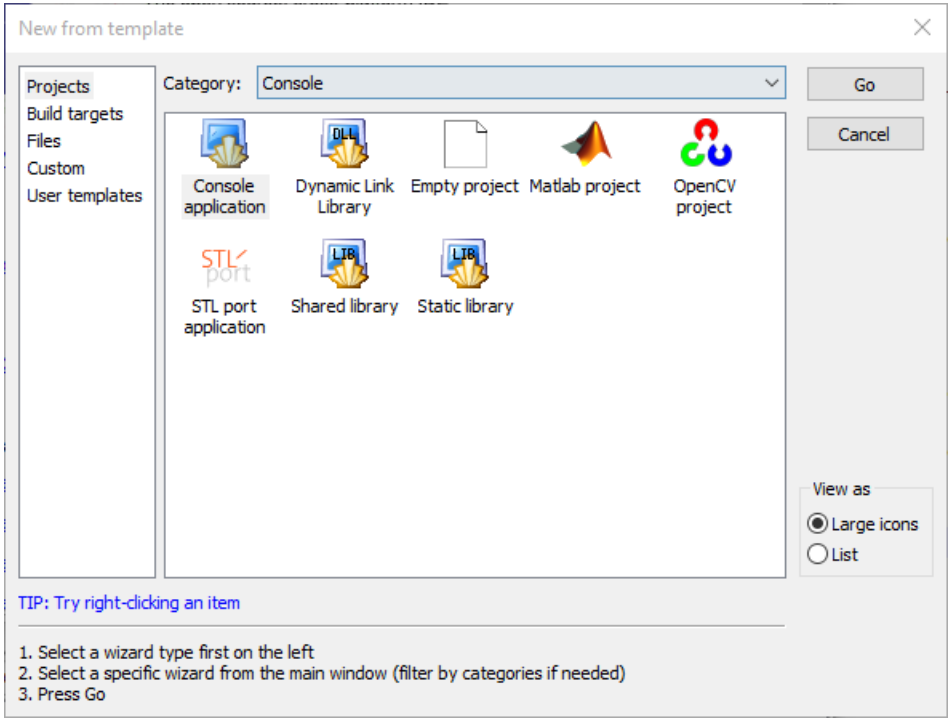
CodeBlocks

- 1. Criação de um projeto



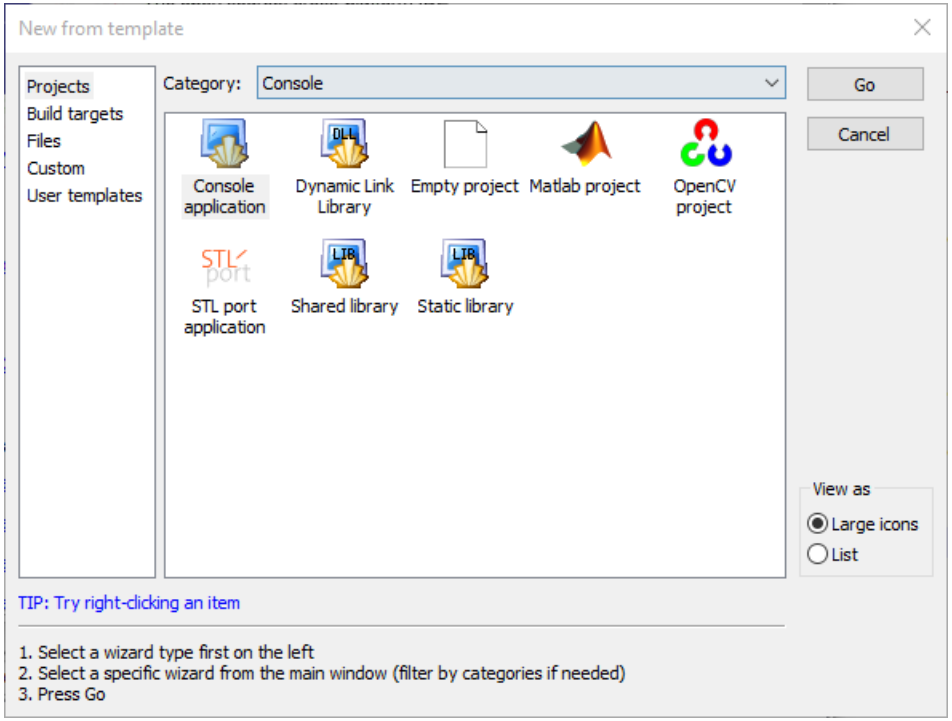
CodeBlocks

- 2. Escolha da categoria *Console*



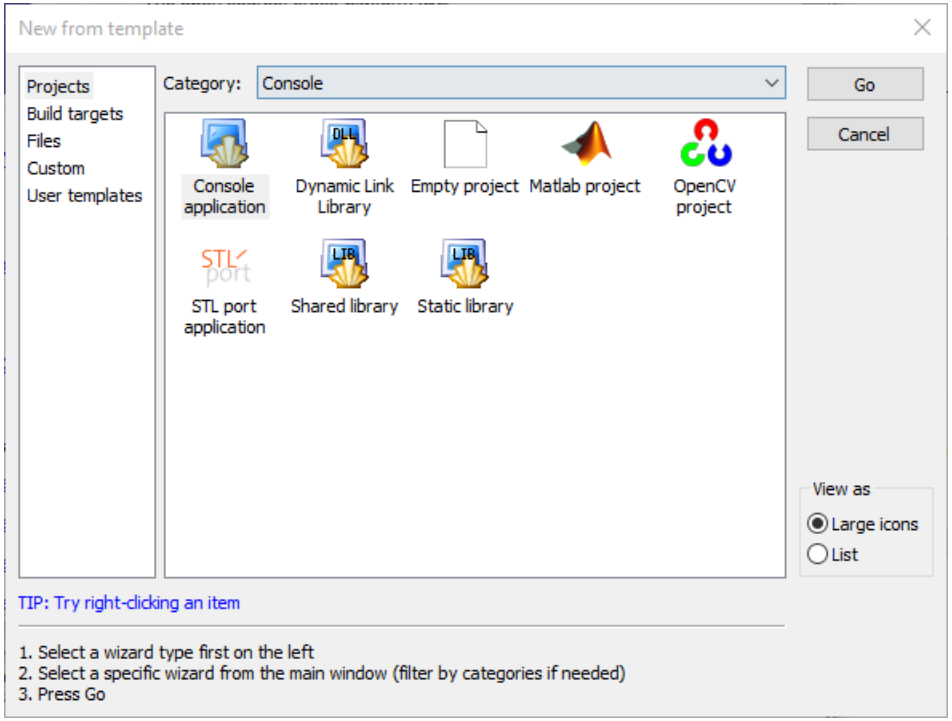
CodeBlocks

- 2. Escolha da categoria *Console*



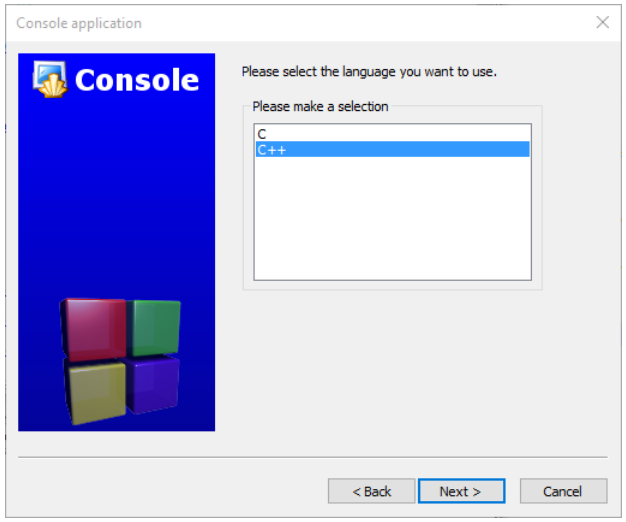
CodeBlocks

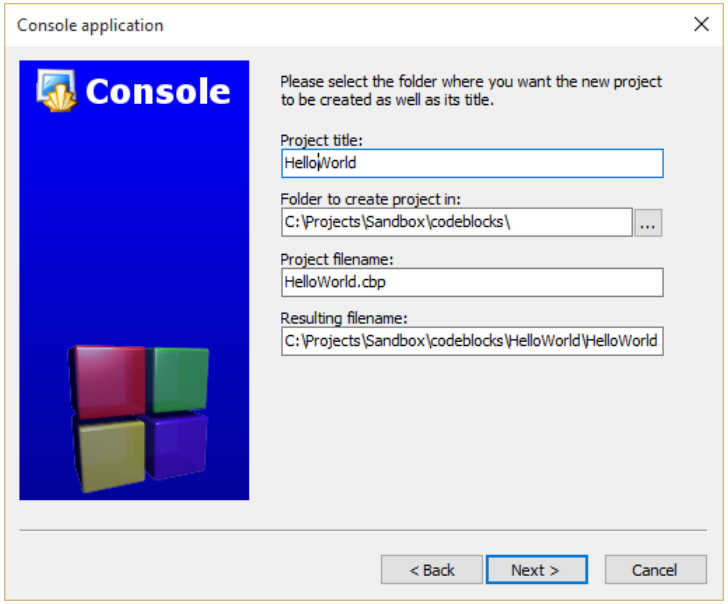
- 2. Escolha da categoria *Console*



CodeBlocks

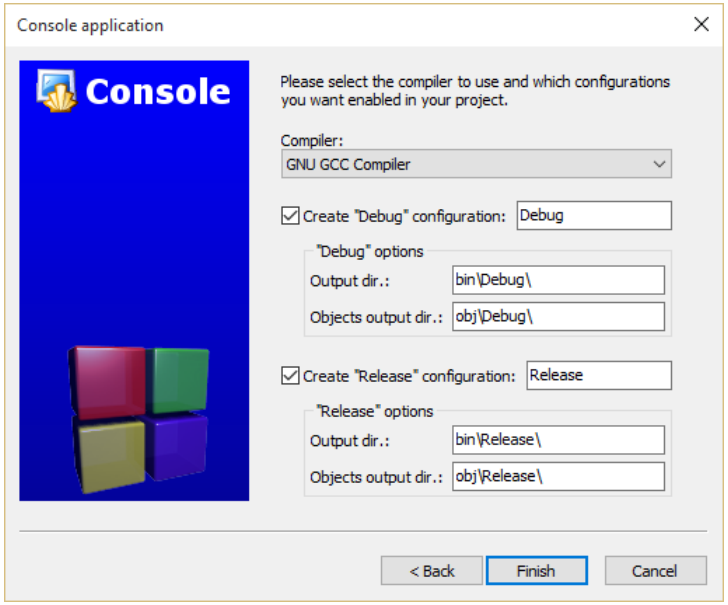
- 3. Seleção da linguagem e nome do projeto





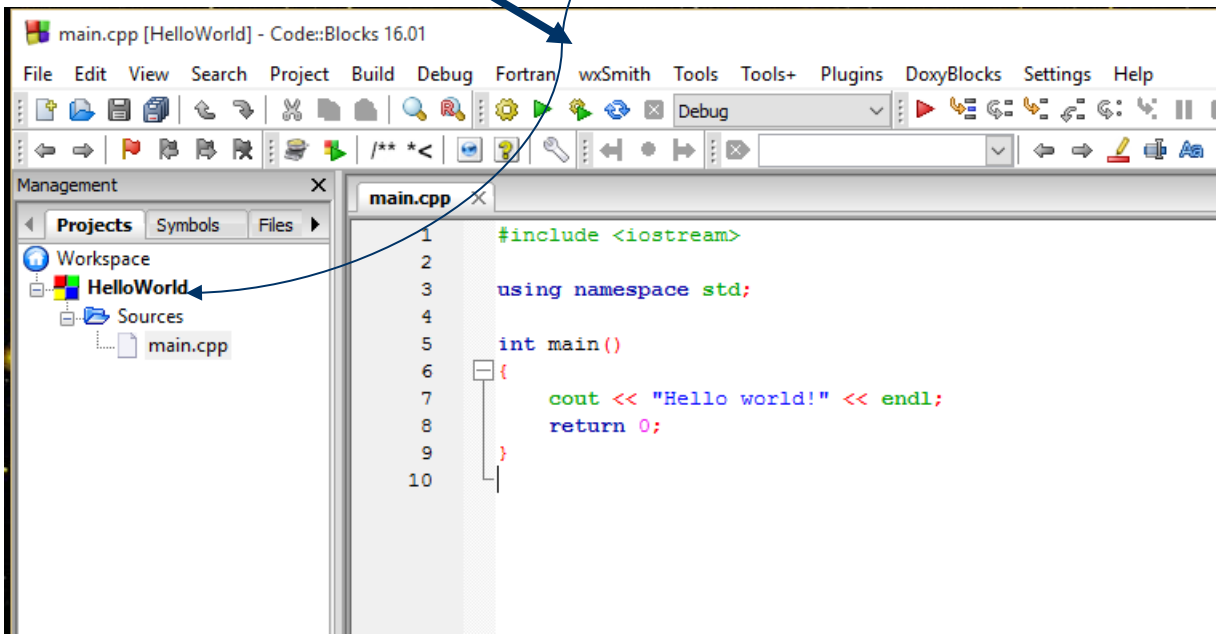
CodeBlocks

- 4. Definição do compilador



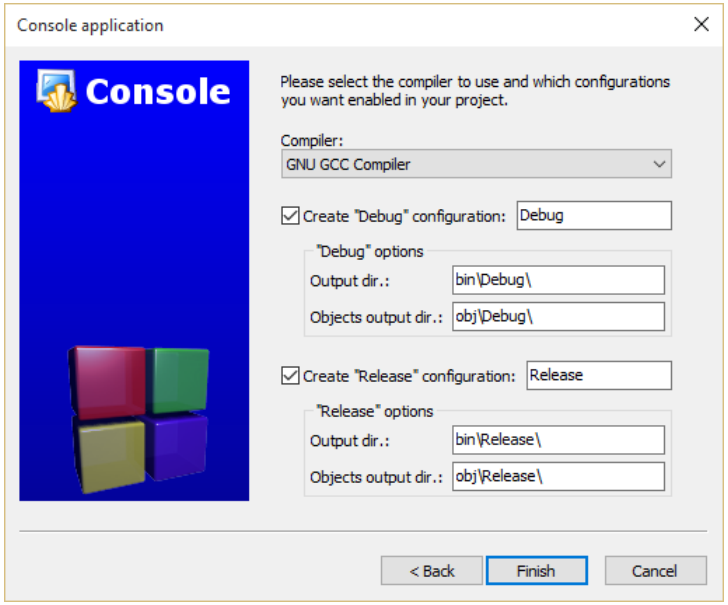
CodeBlocks

- 5. O projeto está criado, basta clicar em *Sources* e *main.cpp*. Clicar no icon  ou pressionar F9 para compilar e executar o programa



CodeBlocks

- 6. Definição do compilador

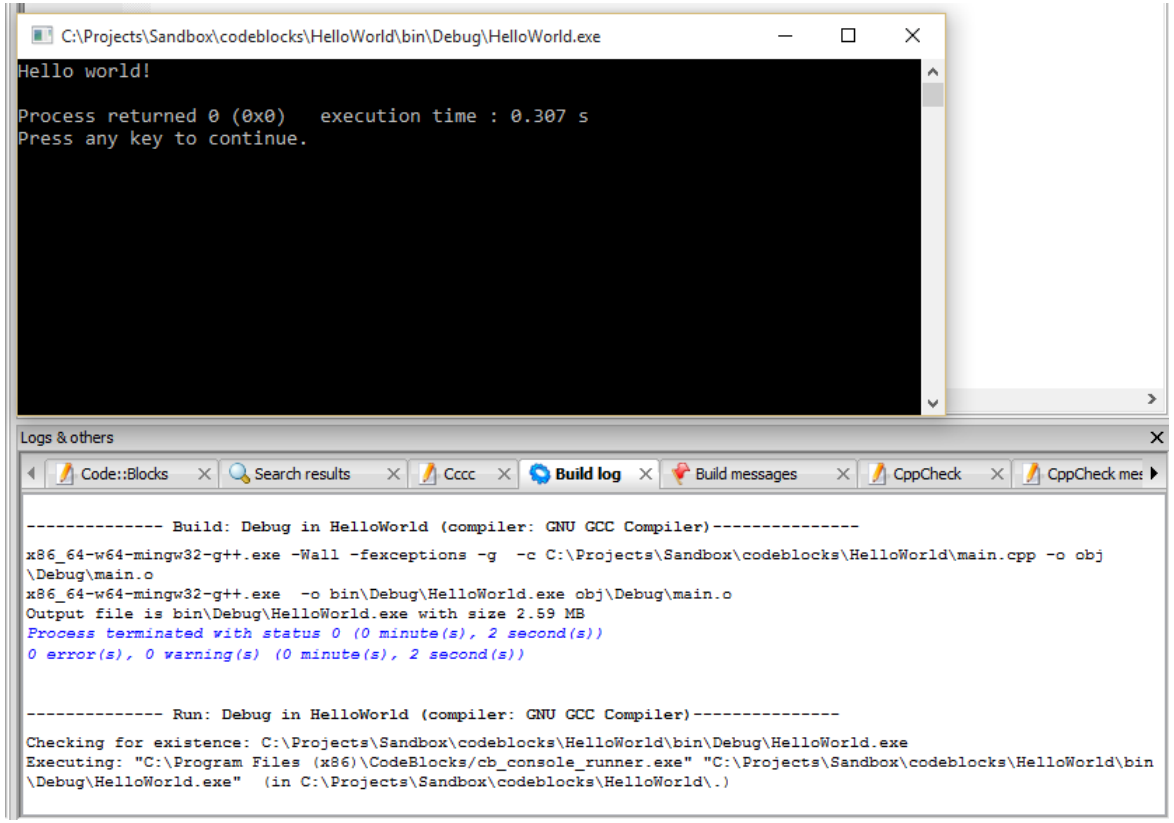


CodeBlocks

- 7. Definição do compilador

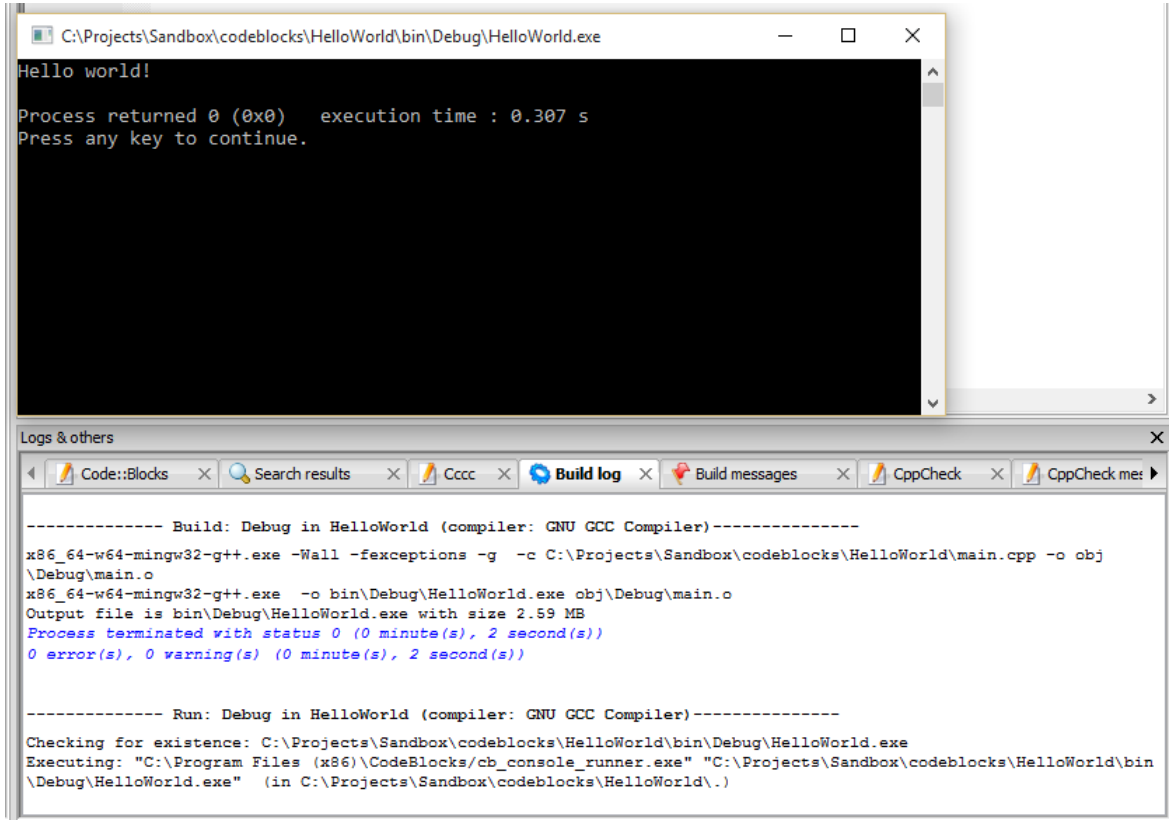
  - Console:

  - Mensagens de compilação:



CodeBlocks

- 8. Definição do compilador
  - Console:
  - Mensagens de compilação:

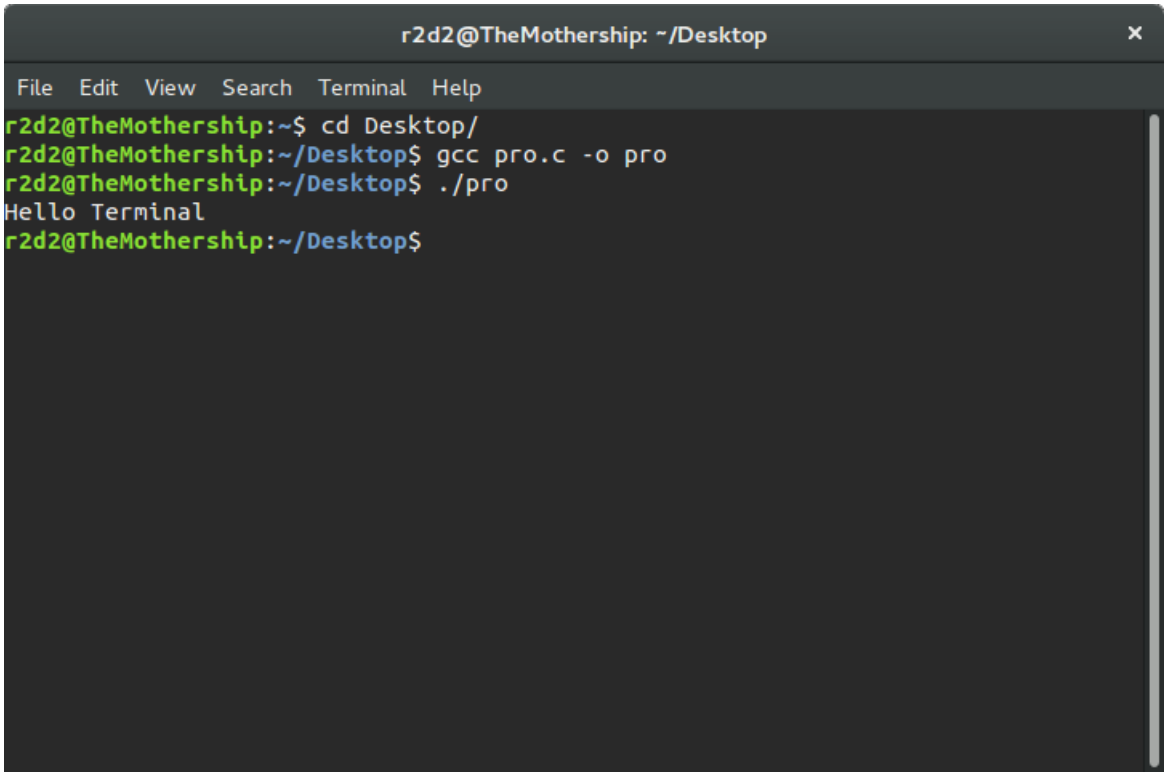


CodeBlocks

- 9. Debugging


```
1   #include <iostream>
2
3   using namespace std;
4
5
6   float RectangleArea1(float L, float W)
7   {
8       return L*W ;
9   }
10
11
12
13  float RectangleArea2(const float L, const float W)
14  {
15      // L=2.0 ;
16      return L*W ;
17  }
18
19
20
21  float RectangleArea3(const float& L, const float& W)
22  {
23      return L*W ;
24  }
25
26
27
28  void RectangleArea4(const float& L, const float& W, float& a
29  {
30      area= L*W ;
31  }
32
33
```

Compilador *gcc*



Primeiro programa em C

```
#include <stdio.h>

int main()
{
    printf("Hello World!\n");
    return (1);
}
```

Linha 1: #include <stdio.h>

- ***#include*** é uma diretiva de pré-processamento
- ***stdio.h*** é uma biblioteca standard do C para input/output
- Os *header files* são incluídos no código fonte pelo pré-processador
- Os *header files* contêm declarações e definições de funções, tipos, e constantes utilizados no código fonte

Linha 2: int main()

- Esta instrução declara a função main
- O programa C pode conter outras funções mas a função main() é obrigatória
- A função main() é o ponto de partida de qualquer programa

Linha 3: {

- A abertura de chaveta indica o início de função
- Todas as funções devem começar com { e termina com }

Linha 4:printf("Hello World!");

- *printf* é uma função de uma biblioteca standard do C que é usada para a escrita de *strings*
- O símbolo "\n" é um modificador de formato especial que indica a mudança de linha
- Esta instrução deve terminar com “;”

Linha 5: return (1);

- A instrução *return* finaliza o programa e neste caso retorna o valor “1”.

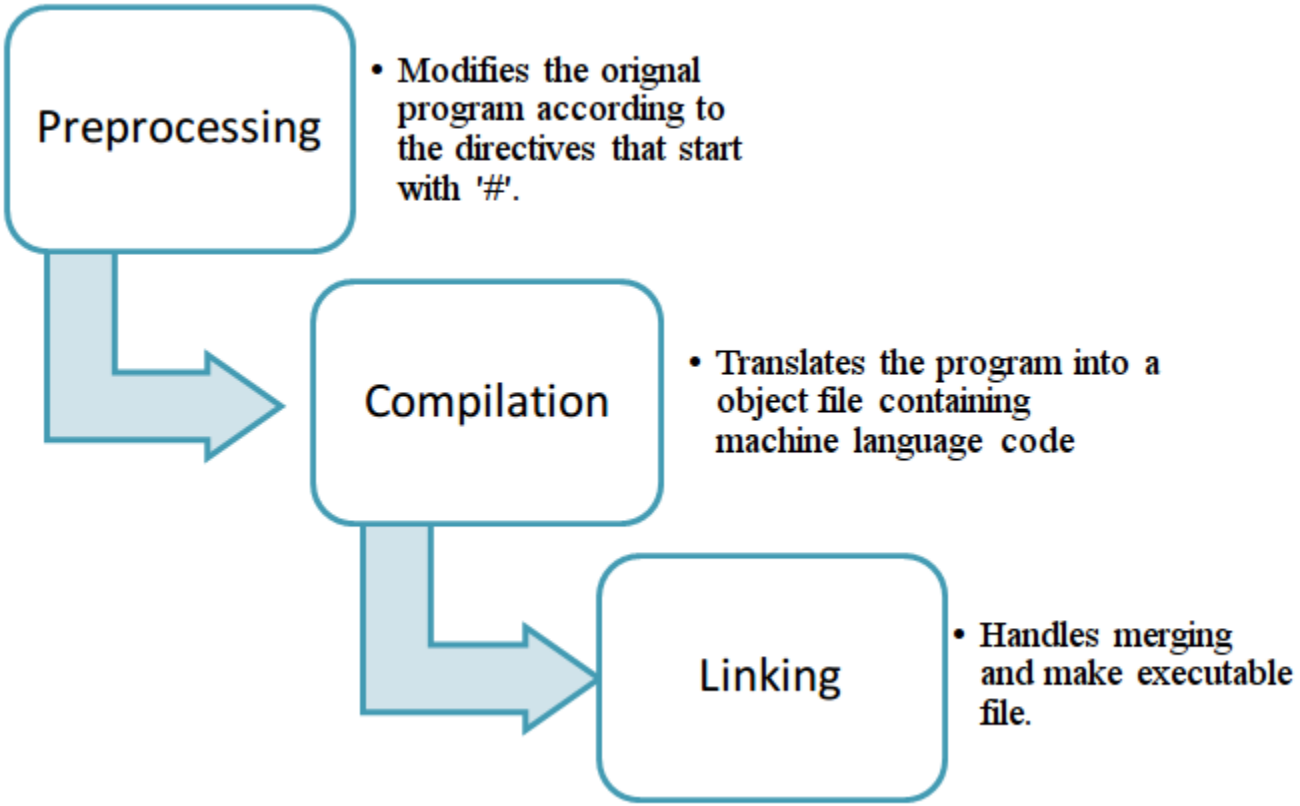
Linha 6: }

- A chaveta indica o fim do programa.

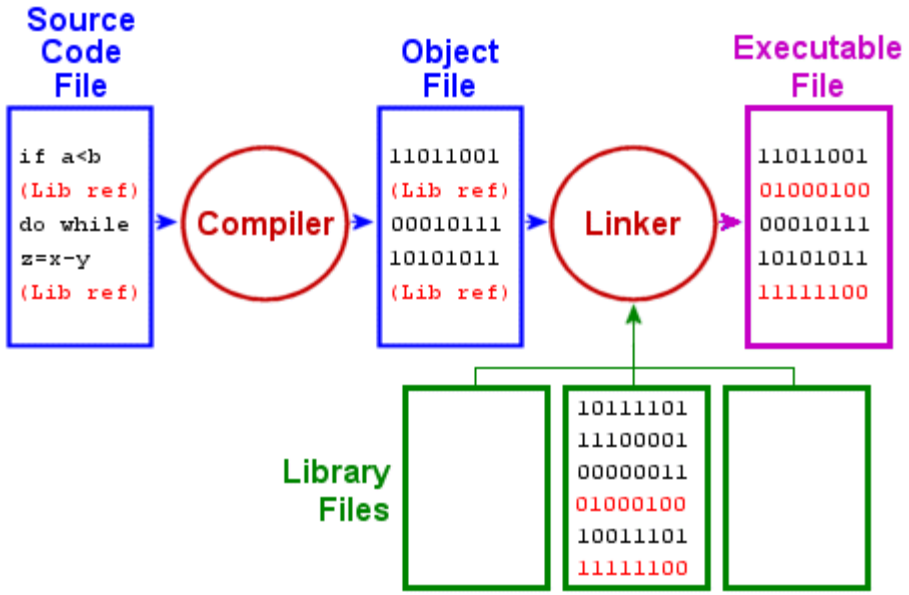
Síntaxe C

- Todas as instruções terminar por ”;”
- O início e o fim dos blocos são indicados por *{* e *}*
- Os comentários são feitos por // ou /* .. */
- É obrigatório a função *main* e a inclusão de *bibliotecas*

Compilação do código fonte



Compilação do código fonte



Compilação do código fonte

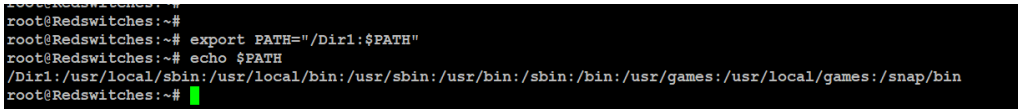
- Compilação :
  - gcc –c helloworld.c
- Compilação e *linkagem*:
  - gcc helloworld.o

- Compilação e *linkagem*:
  - gcc –o helloworld helloworld.c

- Execução:
  - ./helloworld

Compilação do código fonte

- * gcc - É o compilador utilizado
- * -c — Este parâmetro instrui o gcc para compilador o código fonte para ficheiros *object*
- export PATH=$PATH:/path/to/directory



Bibliotecas *standard*

<assert.h>	<complex.h>	<ctype.h>	<errno.h>
<fenv.h>	<float.h>	<inttypes.h>	<iso646.h>
<limits.h>	<locale.h>	<math.h>	<setjmp.h>
<signal.h>	<stdarg.h>	<stdbool.h>	<stddef.h>
<stdint.h>	<stdio.h>	<stdlib.h>	<string.h>
<tgmath.h>	<time.h>	<wchar.h>	<wctype.h>

Exercício 1

- Implementação de um programa que escreve no terminal:

  Hello everybody, my name is ….

Primeiro programa em C

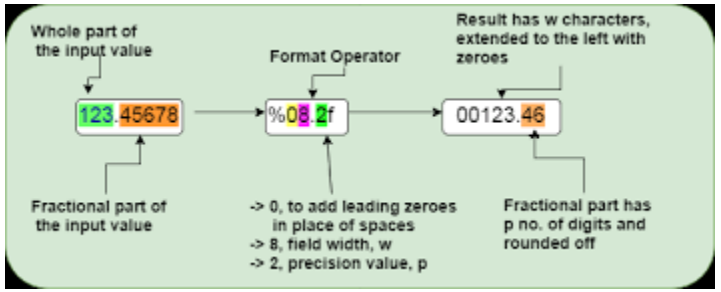
```
#include <stdio.h>

int main()
{
    printf("Hello everybody, my name is….!\n");
    return (1);
}
```

printf () –Especificadores de formato

Code	Format
`%c`	character
`%d`	signed integers
`%E`	scientific notation, with a uppercase "E"
`%f`	floating point
`%s`	a string of characters
`%u`	unsigned integer
`%X`	unsigned hexadecimal, with uppercase letters
`%p`	a pointer
`%%`	a '%' sign
`\n`	New line character
`\"` `\'`	Quotation characters
`\l`	Linefeed character
`\0`	Null ‘character’

Outros detalhes como a precisão podem ser específicados, com a seguinte síntaxe:

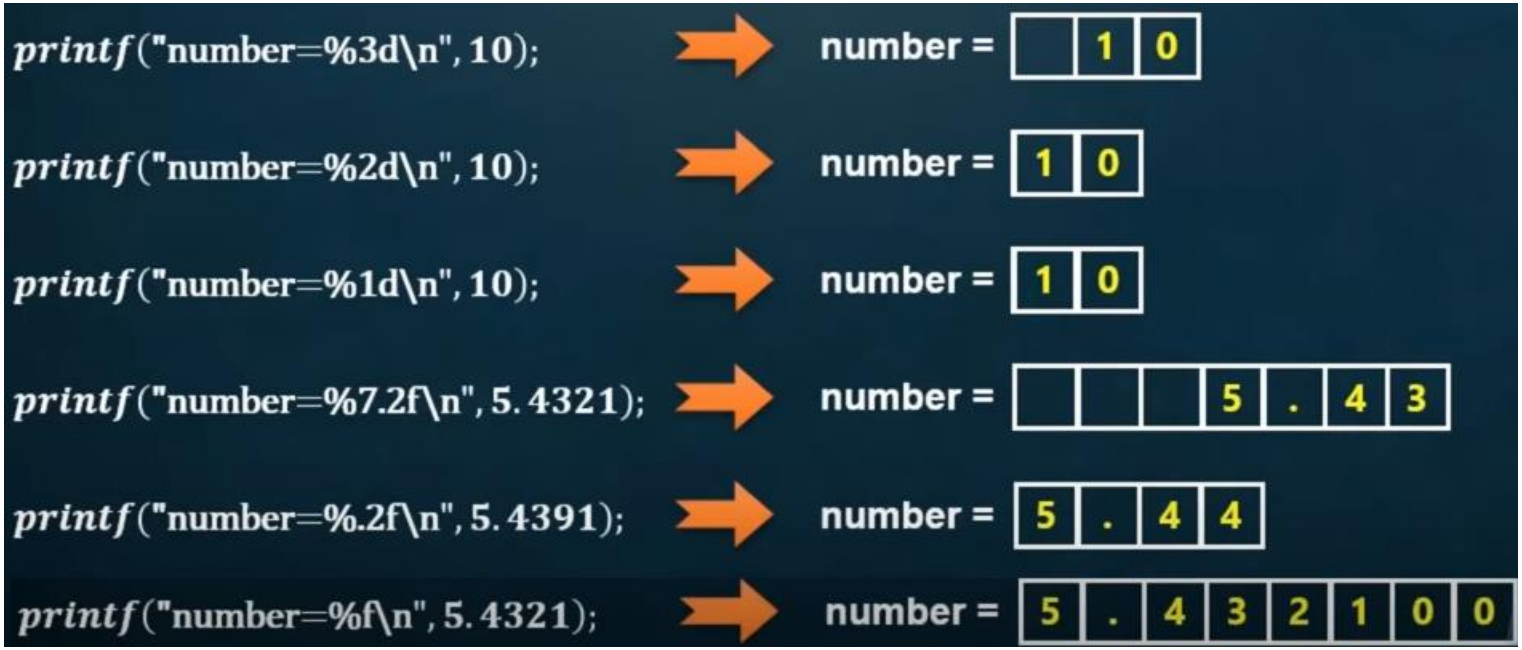
[flags][width][.precision]



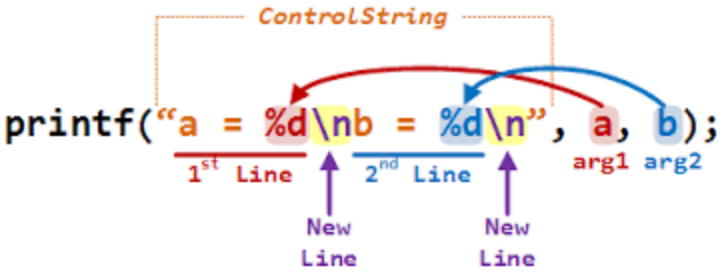
printf () –Especificadores de formato

Outros detalhes como a precisão podem ser específicados, com a seguinte síntaxe:

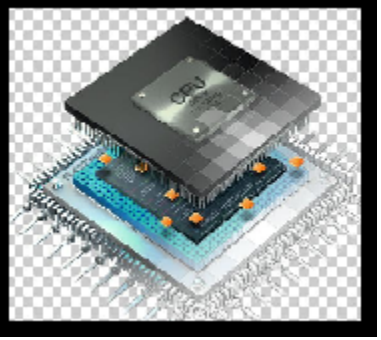
[flags][width][.precision]



printf () –Especificadores de formato




```
printf("siva\n");
printf("%s\n", "siva@meteonic.com");
printf("%s\n", "Microcontrollers");
printf("%s\n", rtos);
printf("Peripheral Interfacing\n");
```

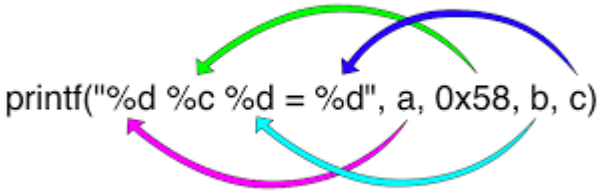


printf () –Especificadores de formato

```
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
```

Laboratório 1

Exercícios sobre o *printf*



Tipos de dados básicos

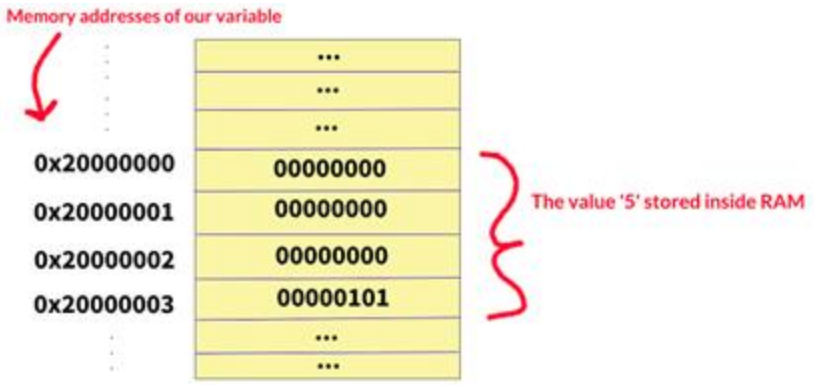
- int – 2 byte integer
- long – 4 byte integer
- float – 2 byte floating point (real)
- double – 4 byte floating point
- char – single byte character
- unsigned char – single unsigned byte

Tipos de dados básicos

Tipo	Número de bits	Formato de leitura com scanf	Intervalo	
			Inicio	Fim
char	8	%c	-128	127
unsigned char	8	%c	0	255
signed char	8	%c	-128	127
int	16	%i	-32.768	32.767
unsigned int	16	%u	0	65.535
signed int	16	%i	-32.768	32.767
short int	16	%hi	-32.768	32.767
unsigned short int	16	%hu	0	65.535
signed short int	16	%hi	-32.768	32.767
long int	32	%li	-2.147.483.648	2.147.483.647
signed long int	32	%li	-2.147.483.648	2.147.483.647
unsigned long int	32	%lu	0	4.294.967.295
float	32	%f	3,4E-38	3,4E+38
double	64	%lf	1,7E-308	1,7E+308
long double	80	%Lf	3,4E-4932	3,4E+4932

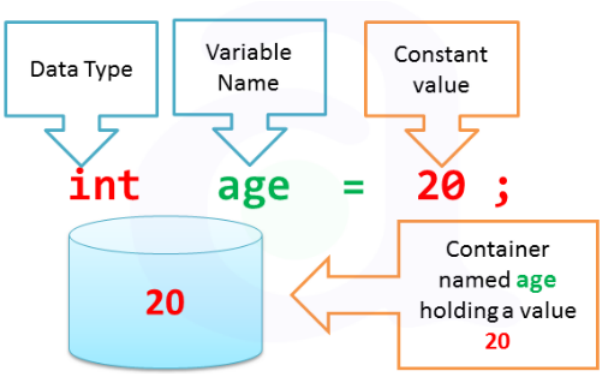
Variáveis

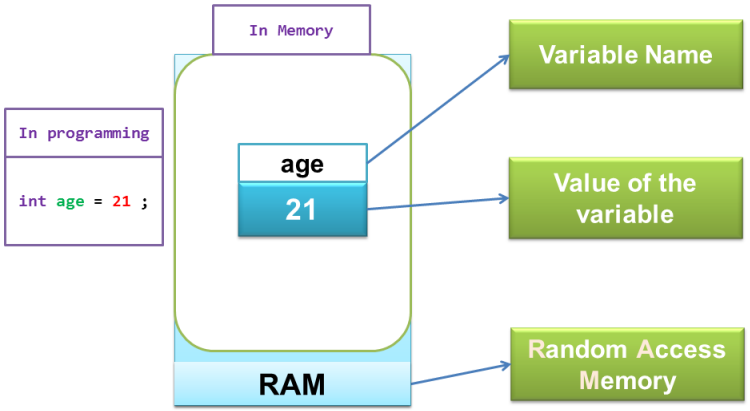
- Espaços de memória reservados que armazenam valores durante a execução de um programa



- Também conhecidas como identificadores

Variáveis






```
#include <stdio.h>

int main()
{
   int age=20;
   return (1);

}
```

```
#include <stdio.h>

int main()
{
   int age;
   age=20;
   return (1);
}
```

Variáveis

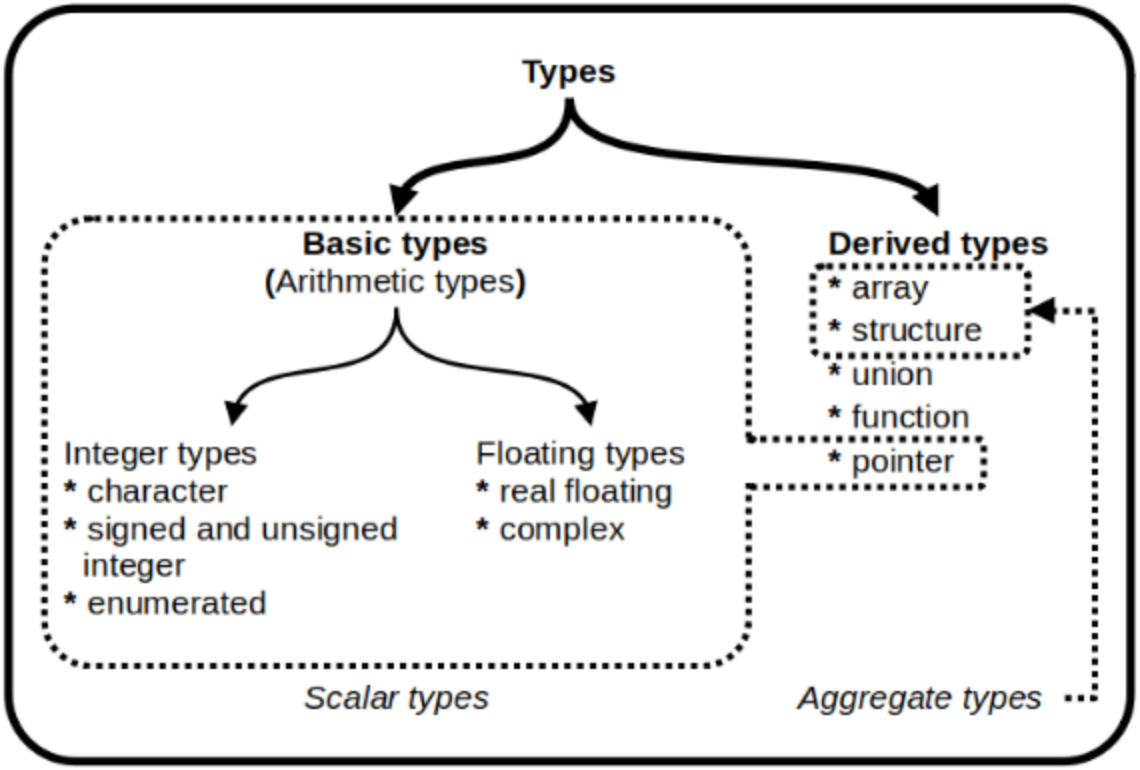
- **Regras para os nomes das variáveis:**
  - Só pode conter letras, digitos ou _
  - O primeiro caracter deve ser uma letra ou _
  - As palavras reservadas do C, não podem ser usadas com identificadores

Variáveis

```
int 2data; //Invalid

float 9test; // Invalid

int abc; //Valid

int Abc; //Valid
```

```
int switch;  //invalid

float for;   //invalid

int goto;   //invalid
```

Tipos de dados básicos



Laboratório 2

Exercícios sobre variáveis

```
int    age   =   20 ;
```

printf () e variáveis

```
#include<stdio.h>

void main()
{
   int a,b;
   float c, d;
   a = 15;
   b = a / 2;
   printf("%d\n", b);
   printf("%3d\n",b);
   printf("%03d\n",b);
   c = 16.3;
   d = c / 3;
   printf("%3.2f\n", d);
   printf(“%4.3f\n”, d);
}
```

Terminal output

```
7
7
007
5.43
5.433
```

Memória

a	15
b	7
c	15.3
d	5.1

Operadores aritméticos

Operador	Ação
-	Subtração
+	Adição
*	Multiplicação
/	Divisão
%	Módulo da divisão (resto)
--	Decremento
++	Incremento

Operadores relacionais/lógicos

Operador	Ação
>	Maior que
>=	Maior ou igual a
<	Menor que
<=	Menor ou igual a
==	Igual a
!=	Diferente de

Operador	Ação
&&	And (E)
||	Or (Ou)
!	Not (Não)

Operadores de incremento/decremento

- Operadores unários(só é necessário uma variável)
  - ++
  - --

Incremento	Decremento
x = x + 1;	x = x - 1;
x++;	x--;

Operadores compostos de atribuição

- São combinações de operadores que simplificam as instruções

a = a + b	a += b
a = a - b	a -= b
a = a * b	a *= b
a = a / b	a /= b
a = a % b	a %= b

Operadores aritméticos - precedência

- A precedência dos operadores aritméticos é a seguinte:

Precedência	Operador	Descrição	Associatividade
Maior precedência	( )	Parêntesis e chamada de funções	Esquerda para direita
	−, + +, − −, !	Operações unárias (operadores usados com um único valor).	Direita para esquerda
	*, /, % +, −	Operações aritméticas	Esquerda para direita
	<<, >>	Operadores de inserção	Esquerda para direita
	<, <=, >, >= ==, ! =	Operações relacionais	Esquerda para direita
	||, &&	Operações lógicas	Esquerda para direita
Menor precedência	=	Operadores de atribuição	Direita para esquerda

Operadores aritméticos – pré/pós incrementação

- Exercício - Qual o resultado das variáveis x, y e z depois da seguinte sequência de operações ?

```
#include <stdio.h>

int main()
{
int x, y, z;
x = y = 10;
z = (x++);
x = -x;
y++;
x = x + y – (z--);
}
```

x	y	z
10	10	-
11	10	10
-11	10	10
-11	11	10
-10	11	9

z=(x++); //Expressão só com 1 elemento, então o parêntesis sem //efeito.

pós-incrementação

y=x++

- primeiro é efetuada a atribuição e depois o incremento

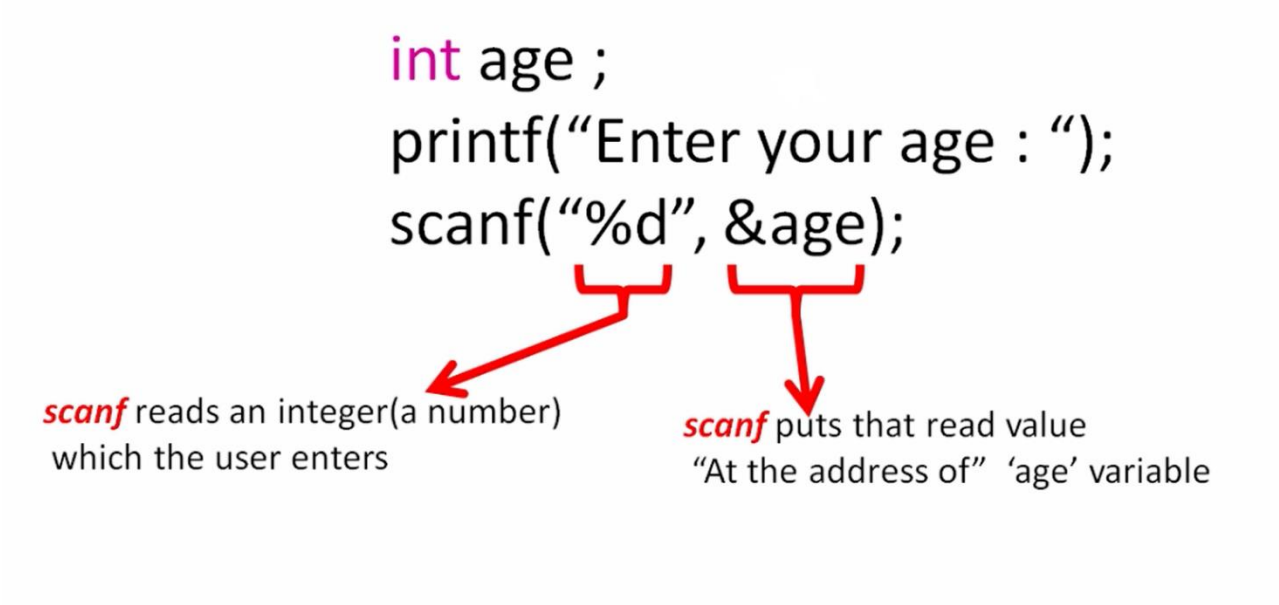
pré-incrementação

y=++x

- primeiro é efetuada o incremento e depois a atribuição

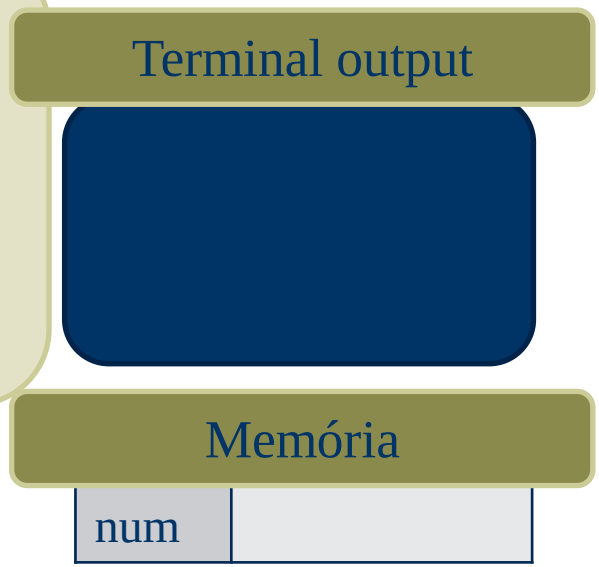
Leitura de dados

- A função *scanf* é utilizada para ler dados provenientes do teclado



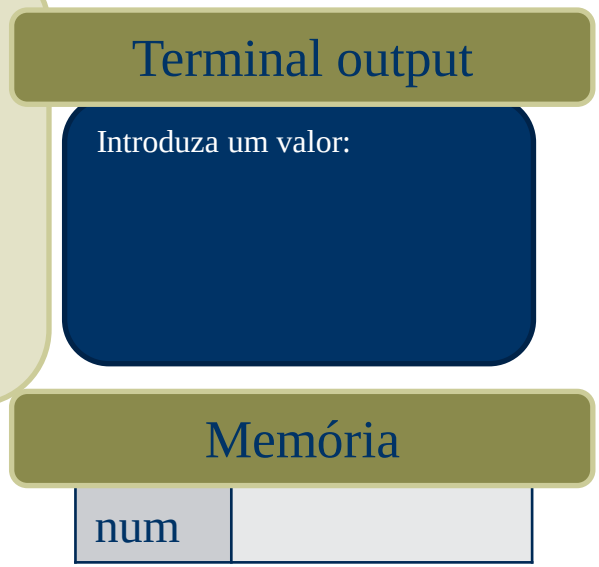
Leitura de dados

```
#include<stdio.h>

void main()
{
   int num;
   printf("Introduza um valor");
   scanf("%d", & num)
   printf("O numero introduzido foi %d", num);
   printf("Fim");
}
```



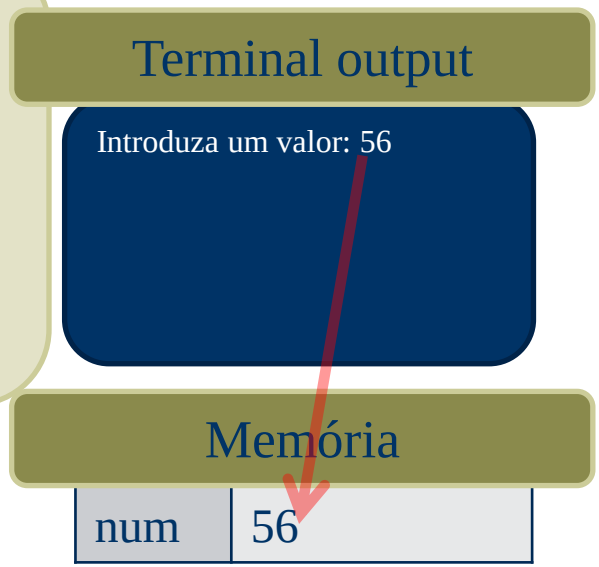
Leitura de dados

```
#include<stdio.h>

void main()
{
   int num;
   printf("Introduza um valor");
   scanf("%d", & num)
   printf("O valor introduzido foi %d", num);
   printf("Fim");
}
```



Leitura de dados

```
#include<stdio.h>

void main()
{
   int num;
   printf(“Introduza um valor”);
   scanf(“%d”, & num);
   printf(“O valor introduzido foi %d”, num);
   printf(“Fim”);
}
```



Leitura de dados

```
#include<stdio.h>

void main()
{
   int num;
   printf(“Introduza um valor”);
   scanf(“%d”, & num);
   printf(“O valor introduzido foi %d”, num);
   printf(“Fim”);
}
```

Terminal output

```
Introduza um valor: 56
O valor  introduzido foi 56
```

Memória

num	56

Laboratório 3

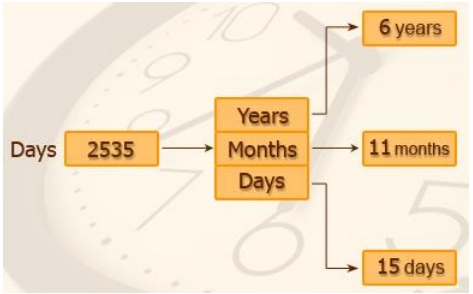
Exercícios sobre o *scanf*

```
scanf("%d",&a);
```

Format string

Address of a variable

Laboratório 3




```c
#include <stdio.h>
int main()
{
  int ndays, y, m, d;
  printf("Input no. of days: ");
  scanf("%d", &ndays);

  y = ndays /365;
  ndays = ndays-(365*y);
  m = ndays /30;
  d = ndays-(m*30);

  printf(" %d Year(s) \n %d Month(s) \n %d Day(s)", y, m, d);

  return 0;
}
```

Introduza um valor: 2535

y = 2535 / 365 = 6,945….= 6

ndays= 2535 - 6*365= 345

m = 345 / 30 = 11,5 = 11

d = 345 – 11*30 = 15

Constantes

- Variáveis com o modificador const não podem ser modificadas por seu programa

```
#include <stdio.h>

int main()
{
   const int age=20;
   return (1);
}
```

Constantes

- A diretiva *#define* permite definir constantes simbólicas que aumentam a legibilidade do código
- Com a diretiva #define é possível realizar substituições parametrizadas
- Não ocupa memória

```c
#include <stdio.h>
#define STUDENT_ID 27
#define COURSE_CODE 502

int main()
{
    printf("Student ID: %d is taking the class %d\n", STUDENT_ID,COURSE_CODE);

    return 0;
}


# Output
Student ID: 27 is taking the class 502
```

Definição de *header files*

- A diretiva *#define* permite definir constantes

#include <stdio.h> para bibliotecas na directoria *default* do sistema

#include "filename.h" para bibliotecas na directoria do código fonte

arith.h

```
float sum (float, float);
float sub (float, float);
float div (float, float);
float mult (float, float);
int mod (int, int);
```

arith.c

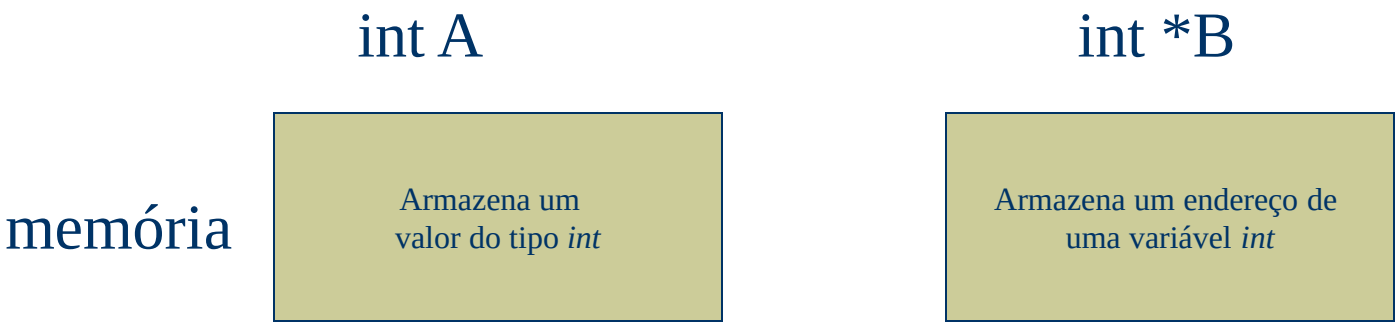
```
#include <stdio.h>
#include "arith.h"   // Include our custom header file

int main()
{
    printf("sum(10, 20)  = %.2f\n", sum(10, 20));
    printf("sub(10, 20)  = %.2f\n", sub(10, 20));
    printf("mult(10, 20) = %.2f\n", mult(10, 20));
    printf("div(10, 20)  = %.2f\n", div(10, 20));
    printf("mod(10, 20)  = %d\n",   mod(10, 20));

    return 0;
}
```

Compilação e *linkagem*:

gcc –o arith arith.c

Introdução aos ponteiros

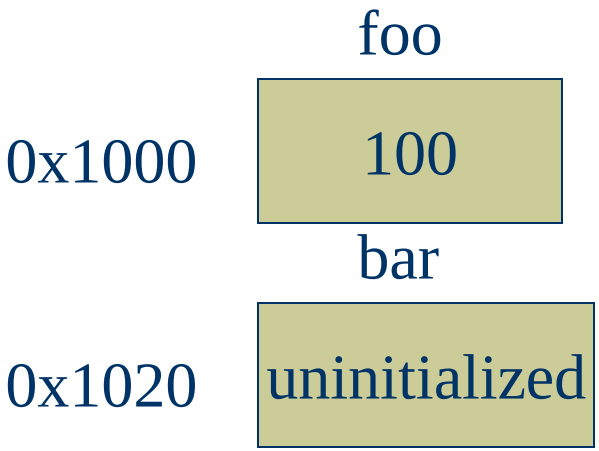
- Variáveis que armazenam endereços de memória
- Utilizados para manipular os valores de outras variáveis



Introdução aos ponteiros

int foo = 100;

*int *bar;*



Qual a diferença ?

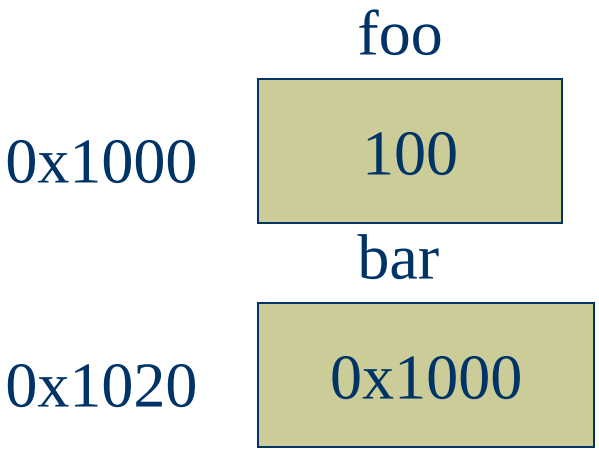
1. foo;
2. &foo;
3. bar;
4. *bar;
5. &bar;

Introdução aos ponteiros

int foo = 100;

*int *bar;*

bar = &foo;



Qual a diferença ?

1. foo;
2. &foo;
3. bar;
4. *bar;
5. &bar;

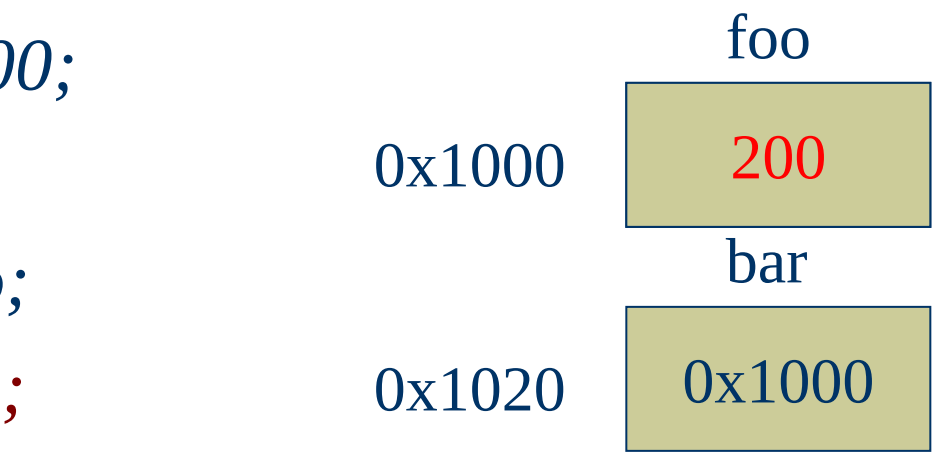
Introdução aos ponteiros

int foo = 100;

*int *bar;*

bar = &foo;

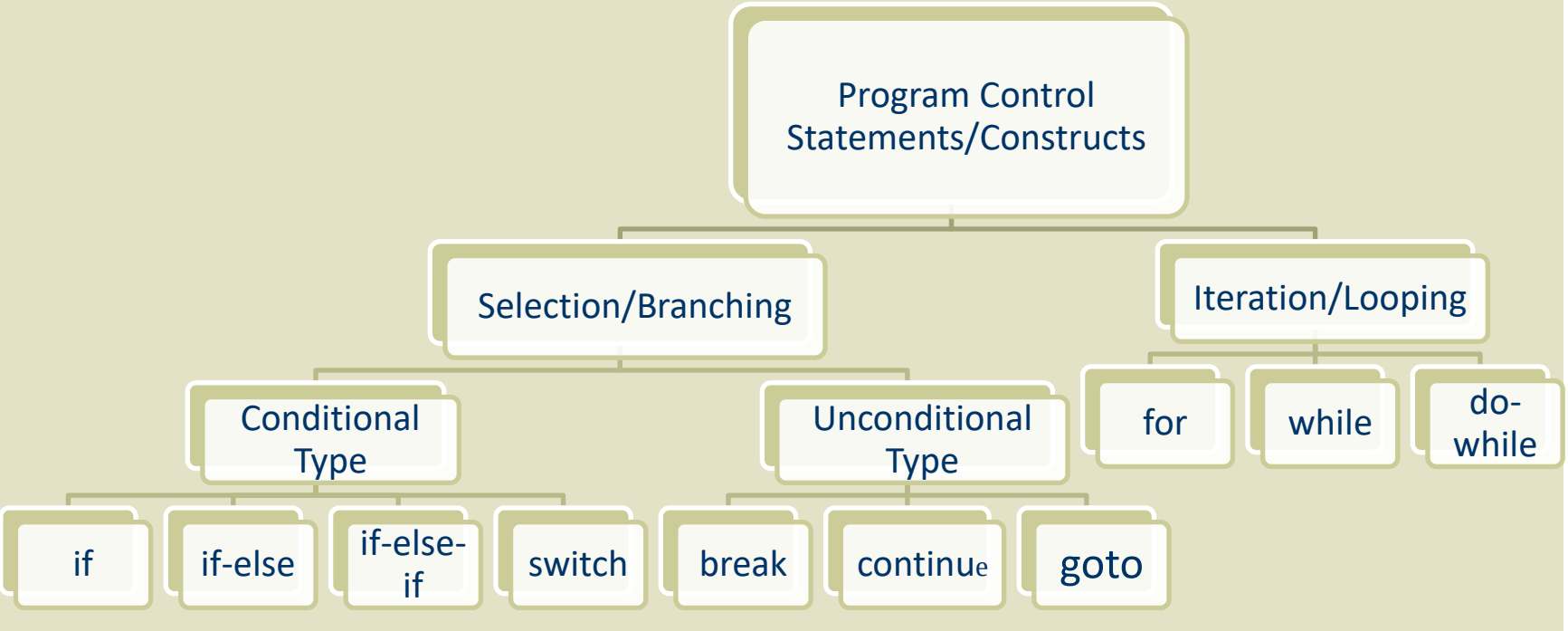
**bar = 200;*



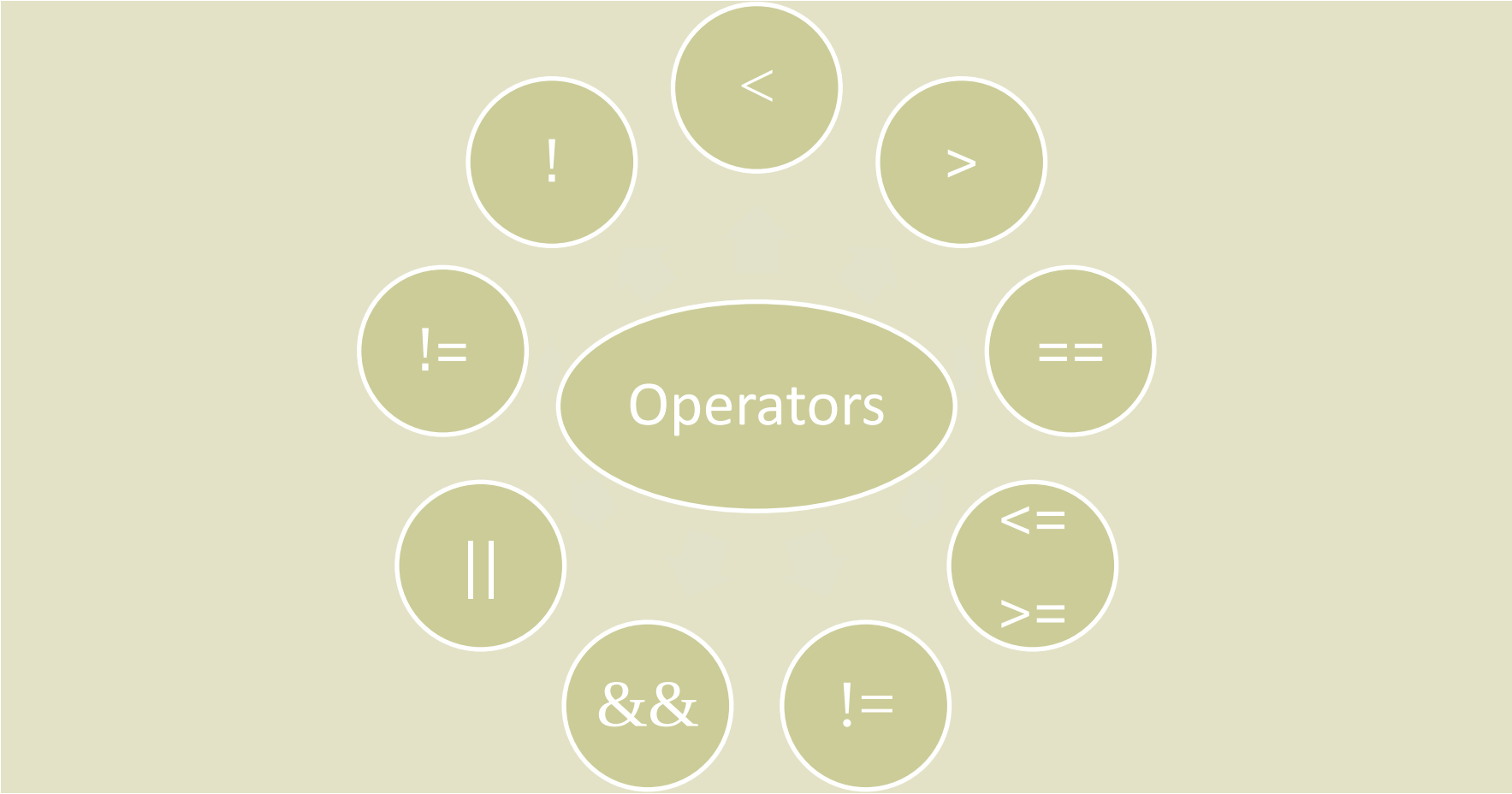
Qual a diferença ?

1. foo;
2. &foo;
3. bar;
4. *bar;
5. &bar;

Estruturas de controlo



Estruturas de controlo

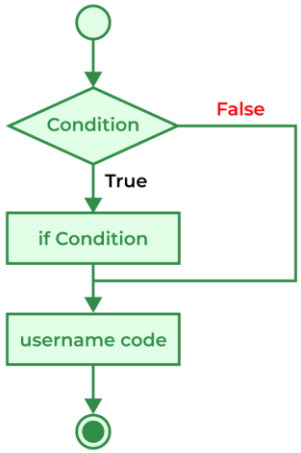


Estruturas de controlo - *if*

- 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*

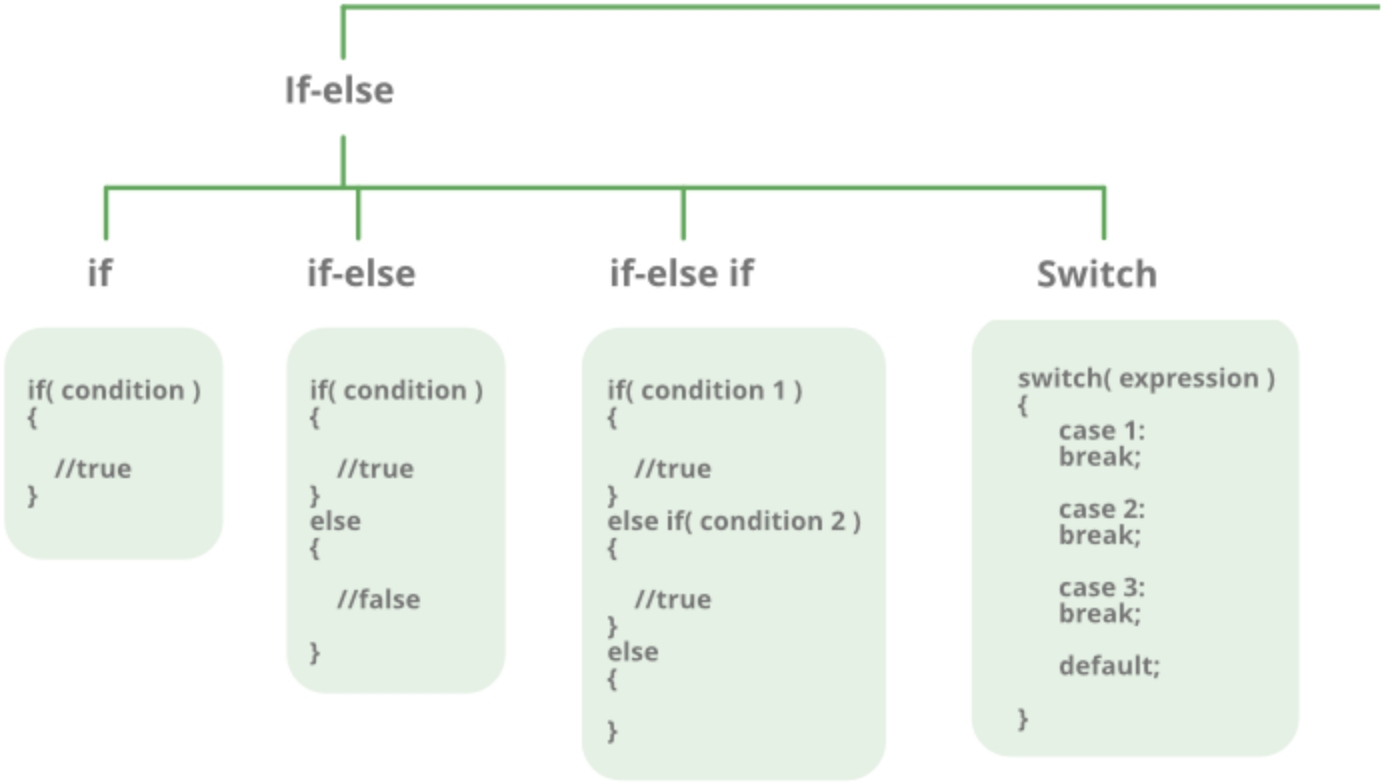


Tabela ASCII

```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
```

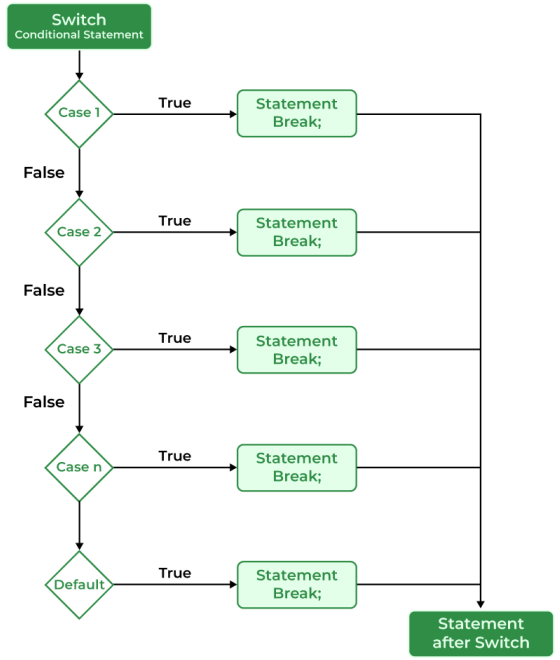
```c
#include<stdio.h>
void main ()
{

  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*

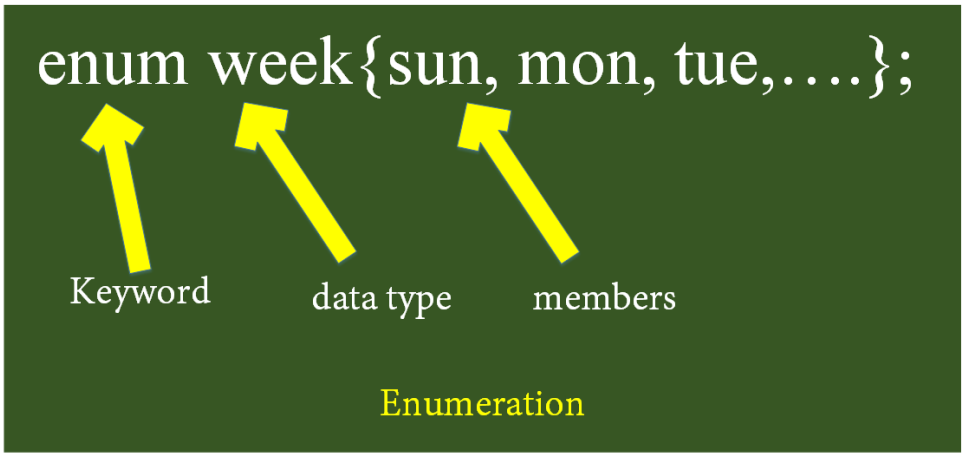
- Estrutura de controlo que testa uma expressão e compara-a a múltiplas opções e executa o bloco da opção correcta

```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;
    }
```



Enumerações

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

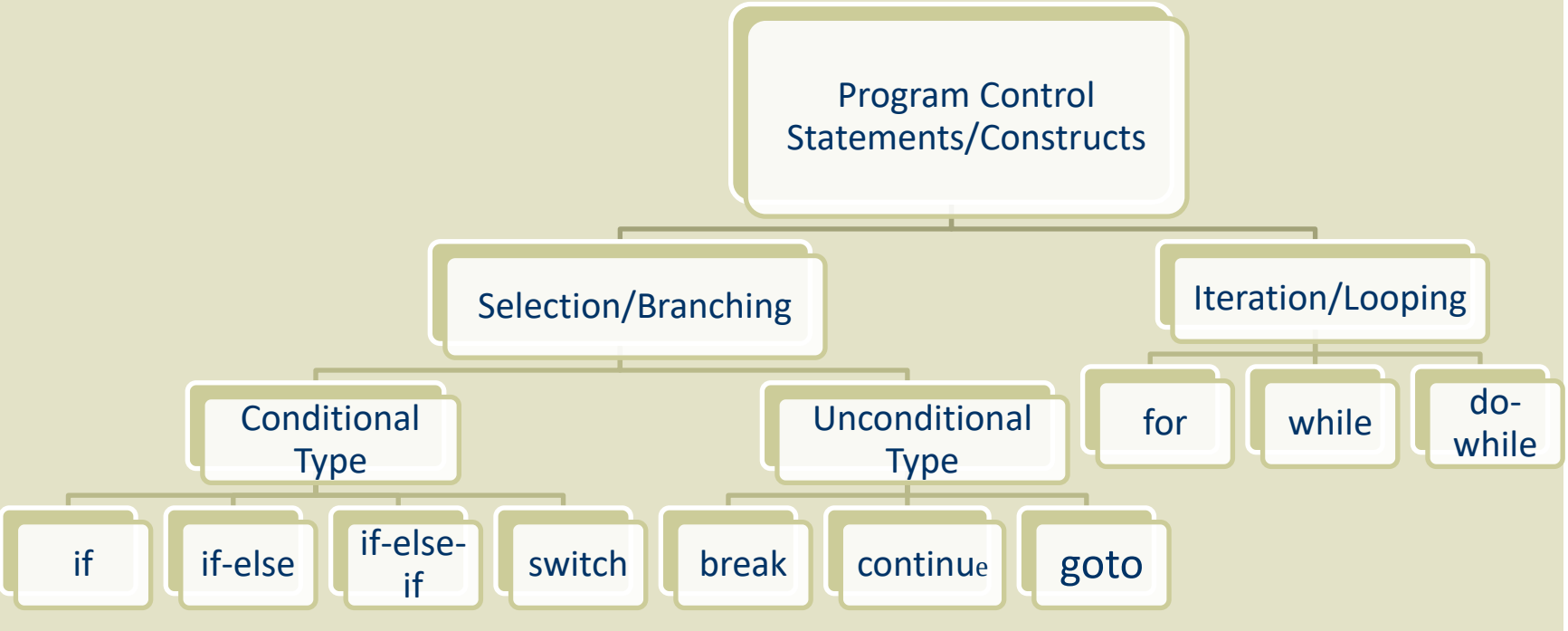


Enumerações

```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;
}
```

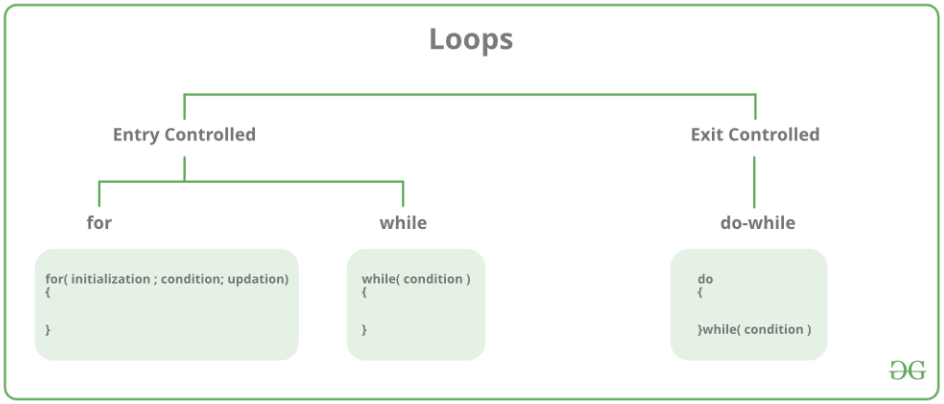
- Uma variável do tipo enumerado, pode assumir o descritivo ou o índice definido
- Só a escrita do índice pode ser efetuada
- Caso não seja definido qualquer índice é assumido o “0” no primeiro elemento
- Caso algum elemento não tenha índice, assume o índice anterior + 1

Estruturas de iterativas

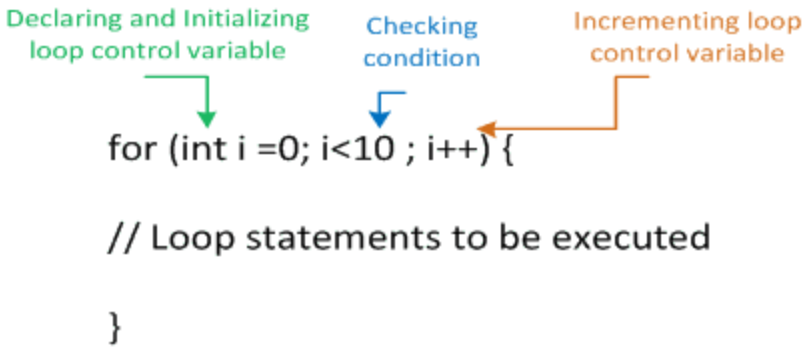


Estruturas iterativas

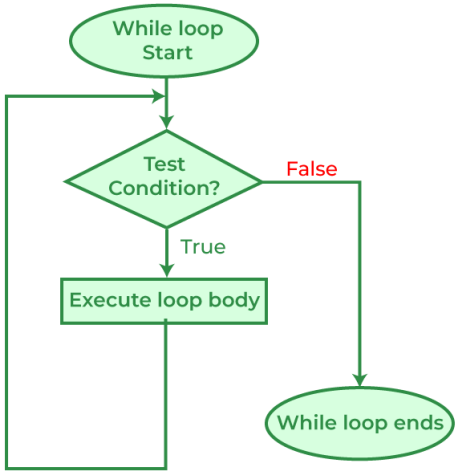
- Os *loops* são utilizados para a repetir o mesmo bloco de código até que uma condição se verifique



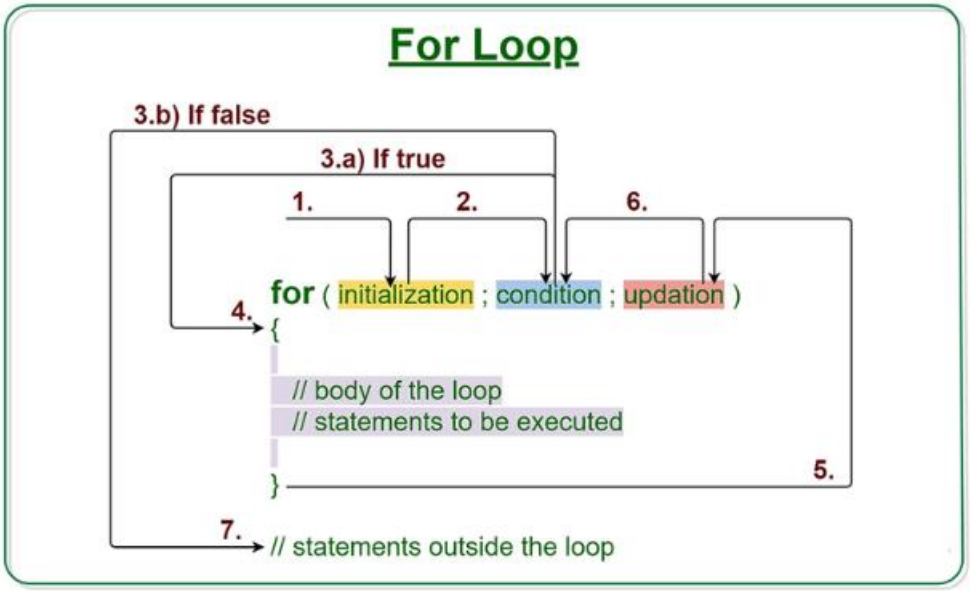
Estruturas iterativas - *for*

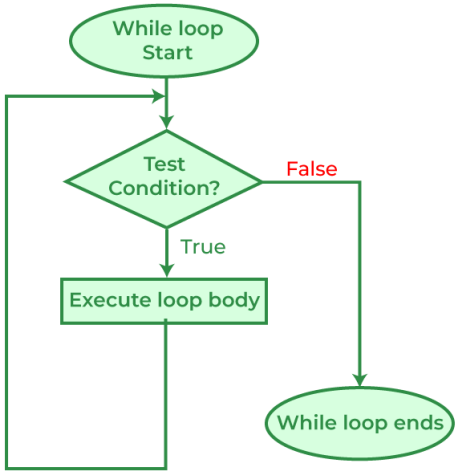



```c
#include <stdio.h>

int main () {

   int a;

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

   return 0;
}
```

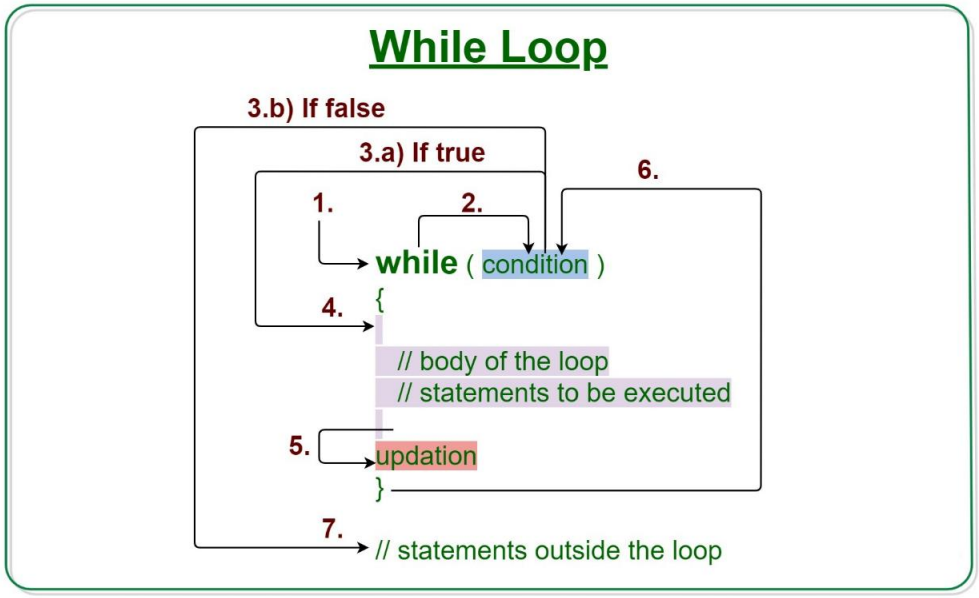


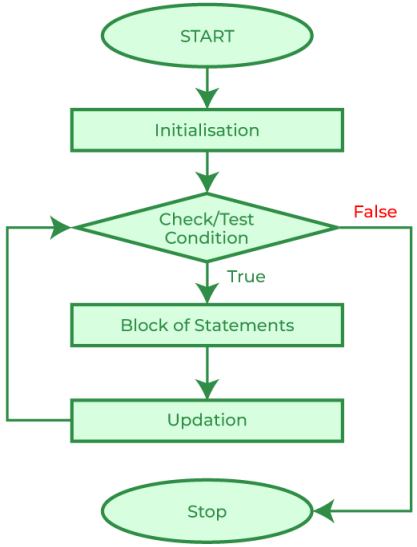
Estruturas iterativas - *for*



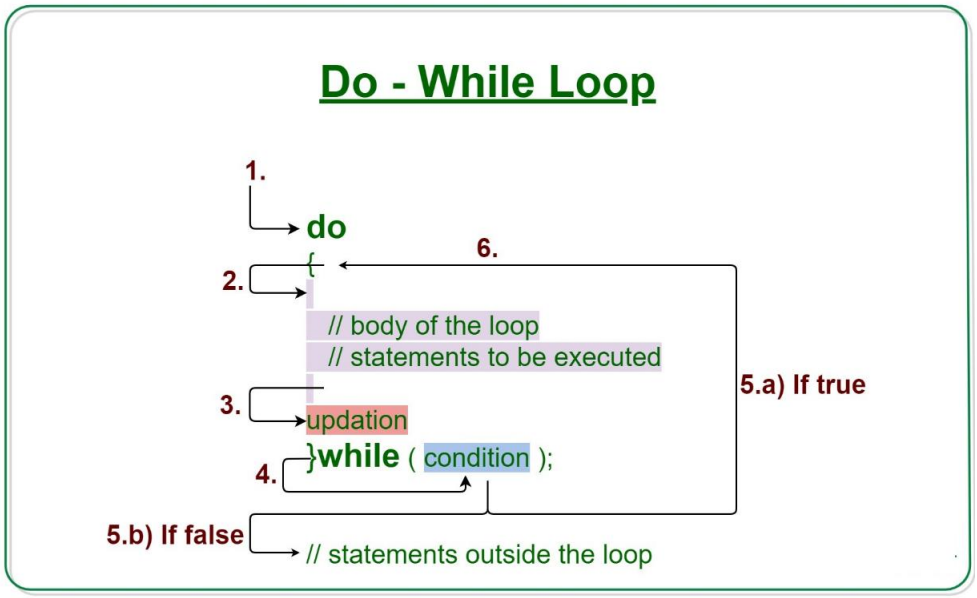


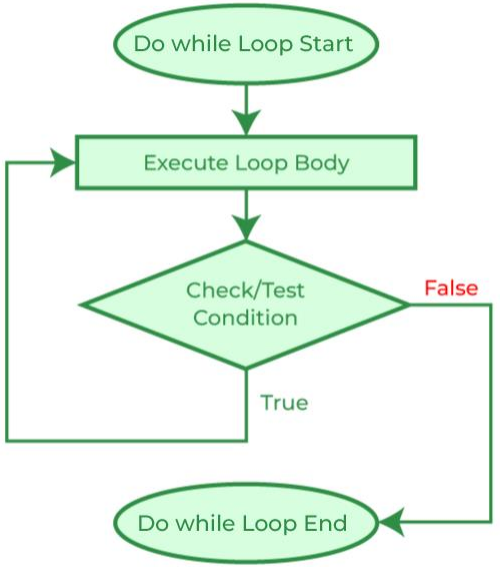
Estruturas iterativas – *while*





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;
}
```

Debugging/depuração

Arrays/vectores

Arrays/vectores

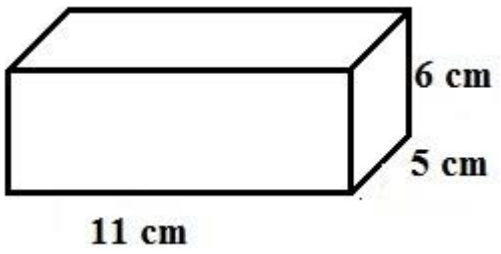
Funções

Alocação de memória

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];
```

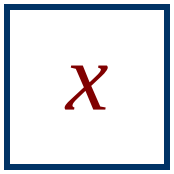
- Every char of name can be accessed as name[i]
- Arrays are indexed from 0 so the first character in a string is `string[0]`

Sample program

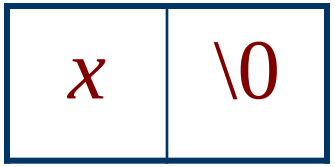
```
int main()
{
    char name[40];
    printf("what is your name?");
    scanf("%s"  name);
    printf("hello  %s.\n"  name);
    return 0;
}
```

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*

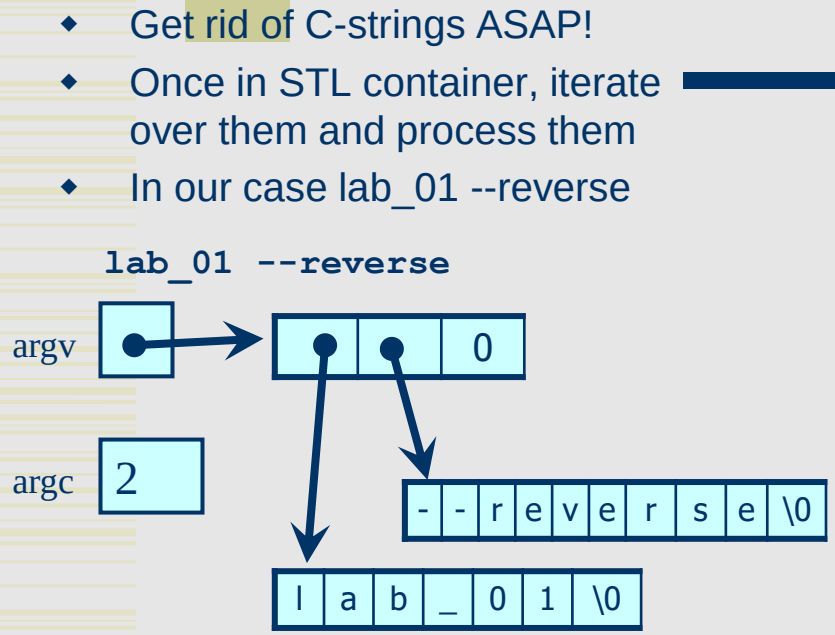
```
do
{… body… }
while (expression);
```

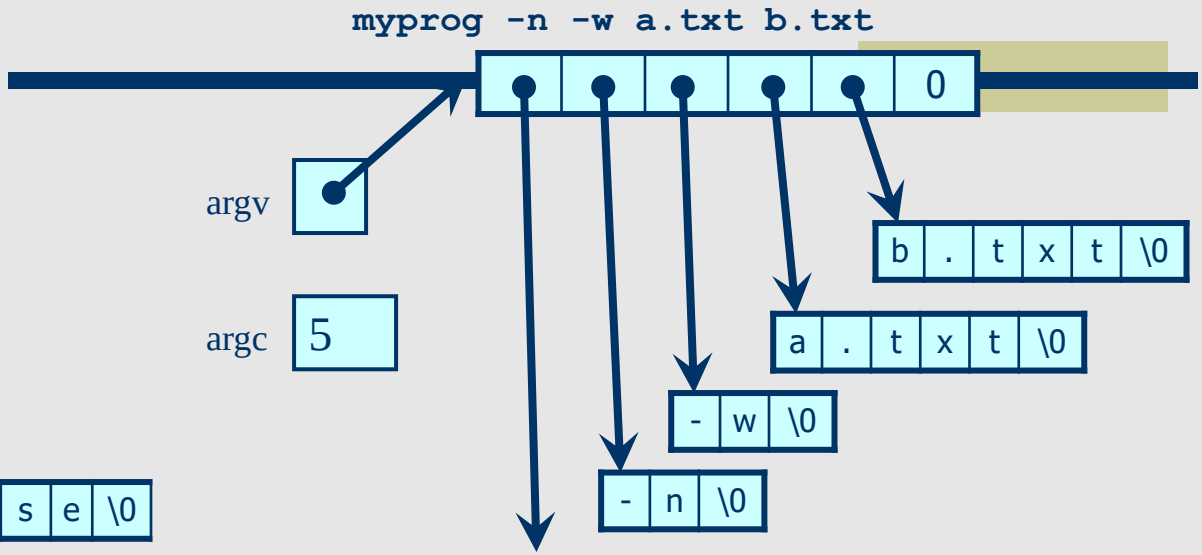
Example

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

Coding: Processing cmd args

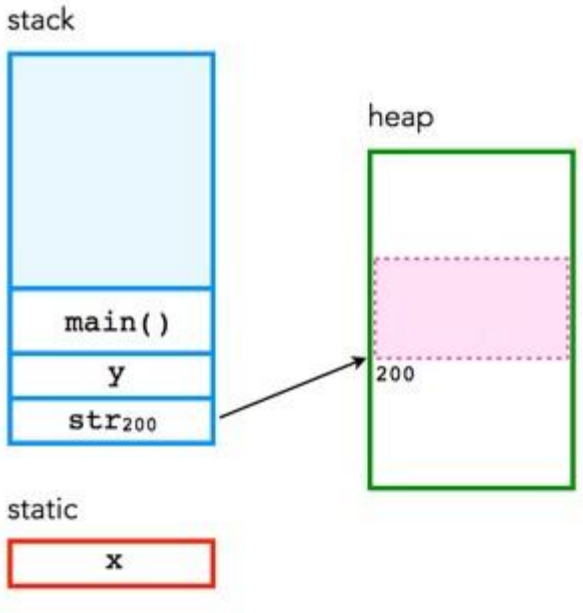
- Get rid of C-strings ASAP!
- Once in STL container, iterate over them and process them
- In our case lab_01 --reverse





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).