

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

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 operativo UNIX
- ◆ Baseada em funções, *weakly typed*, sintaxe 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 transfer from. |
| 1. Checkings to Savings
| 2. Savings to Checkings
|-----|
1
Enter transfer amount: $1000
Your updated Checkings balance is: $1700

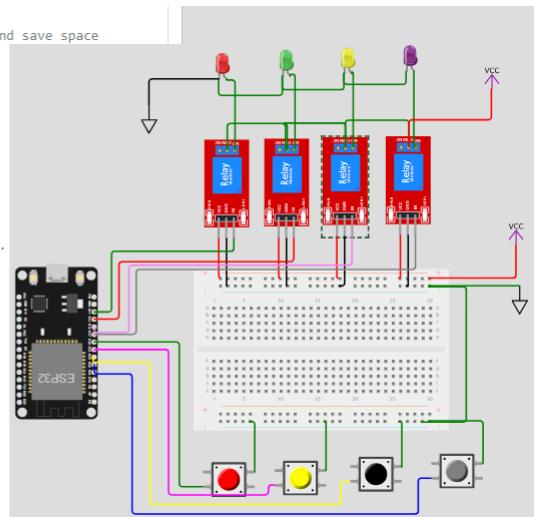
|What would you like to do? |
| 1. Deposit
| 2. Withdraw
| 3. Check Balance
| 4. Transfer Money
| 5. [Exit]
|-----|
3
|     Check Account Balance for? |
| 1. Checking
| 2. Savings
|-----|
2
Your current Savings balance is $1000
```

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

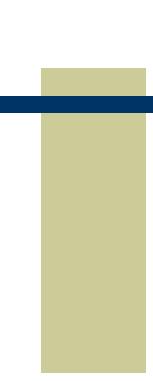
100 % <
Results Messages
HighestInvoice
1 8999.00
```

Aplicação da linguagem C

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

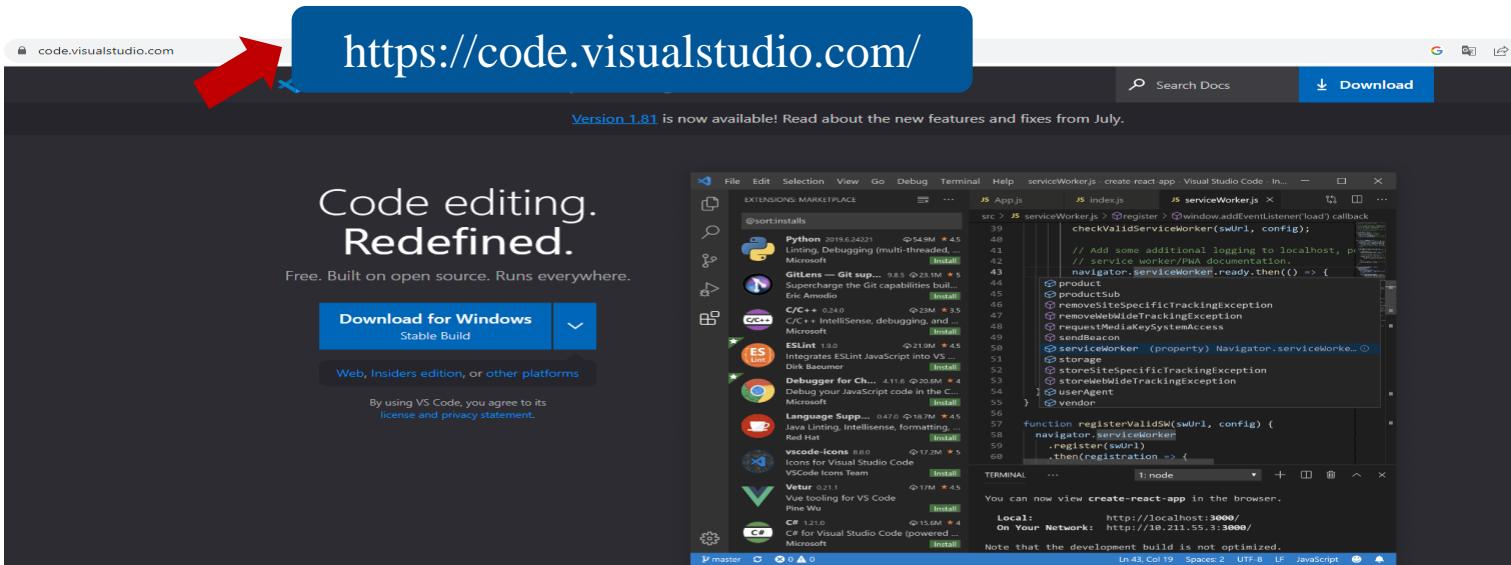
- ✓ C/C++ for Visual Studio Code
- ✓ C/C++ Extension Pack
- ✓ Code Runner - configurar a extensão para Code Runner In Terminal

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”

The screenshot shows the official Visual Studio Code download page at code.visualstudio.com/Download. The page features a dark header with the Visual Studio Code logo, navigation links (Docs, Updates, Blog, API, Extensions, FAQ, Learn), a search bar, and a prominent 'Download' button. Below the header, there are three main download sections: Windows, Linux, and macOS. Each section includes a platform icon (Windows logo, Tux logo, Apple logo), a large download button, and smaller buttons for different file formats (e.g., User Installer, System Installer, CLI, .deb, .rpm, .tar.gz, Snap). At the bottom of the page, there are promotional links for the Insiders build and vscode.dev, and a 'Get previous versions' button.

code.visualstudio.com/Download

Visual Studio Code Docs Updates Blog API Extensions FAQ Learn Search Docs Download

Windows 10, 11

Windows

User Installer x64 x86 Arm64
System Installer x64 x86 Arm64
.zip x64 x86 Arm64
CLI x64 x86 Arm64

.deb x64 Arm32 Arm64
.rpm x64 Arm32 Arm64
.tar.gz x64 Arm32 Arm64
Snap Snap Store
CLI x64 Arm32 Arm64

.zip Intel chip Apple silicon Universal
CLI Intel chip Apple silicon

macOS 10.11+

By downloading and using Visual Studio Code, you agree to the [license terms](#) and [privacy statement](#).

Want new features sooner?
Get the [Insiders build](#) instead.

Use vscode.dev for quick edits online!
GitHub, Azure Repos, and local files.

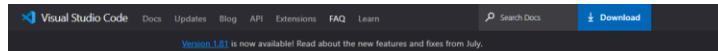
Get previous versions

Compilador C para Windows

Download e instalação do VSCode

Compilador C para Windows

Download e instalação do VSCode



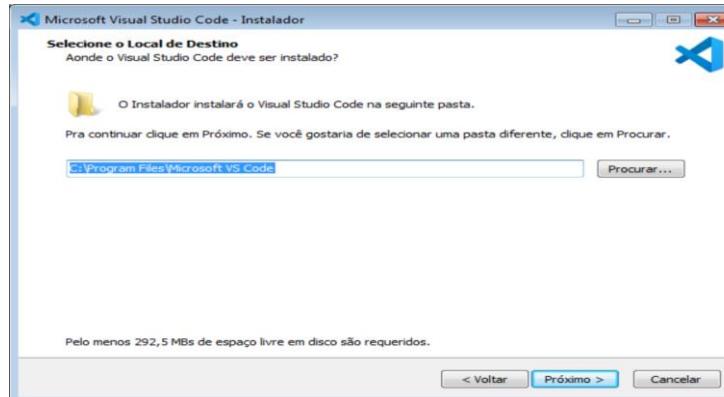
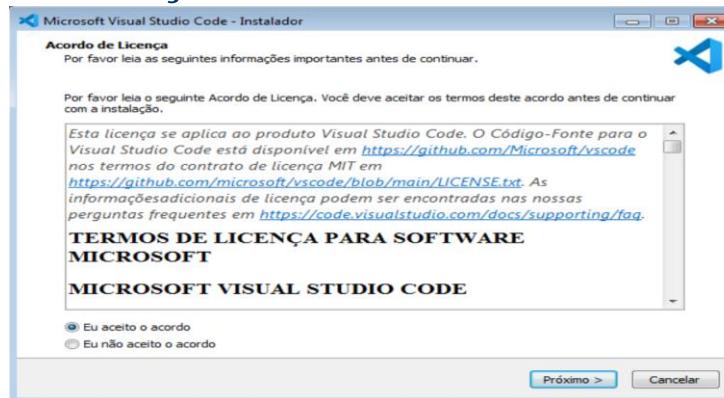
A screenshot of the 'Visual Studio Code FAQ' page. The sidebar on the left is identical to the previous screenshot. The main content area is titled 'Visual Studio Code FAQ'. It includes a section for 'What is the difference between Visual Studio Code and Visual Studio IDE?' and another for 'Which OSs are supported?'. A red box highlights the 'Release notes' link under the FAQ section. To the right, there's a sidebar titled 'IN THIS ARTICLE' with various links related to VS Code.

A screenshot of the 'April 2022 (version 1.67)' update details page. The sidebar on the left is identical to the previous screenshots. The main content area starts with a welcome message: 'Welcome to the April 2022 release of Visual Studio Code. There are many updates in this version that we hope you'll like, some of the key improvements include:' followed by a bulleted list of changes. A large red circle with the number '1' is overlaid on the welcome message. The update details also mention 'If you'd like to read these release notes online, go to [Updates](#) on [code.visualstudio.com](#)'. At the bottom right, there are links for 'Subscribe' and 'Ask questions'.

Compilador C para Windows

Download e instalação do VSCode

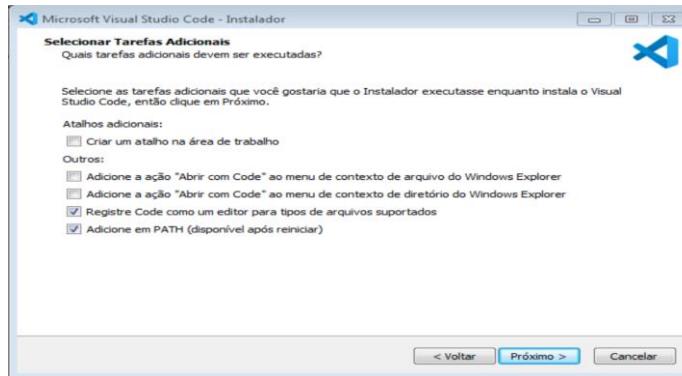
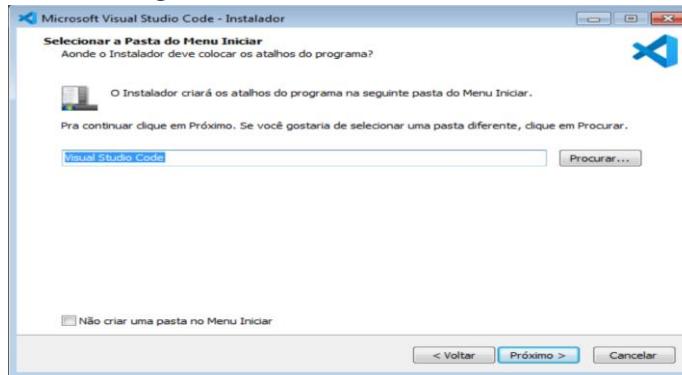
Passo 2: Instalação do executável



Compilador C para Windows

Download e instalação do VSCode

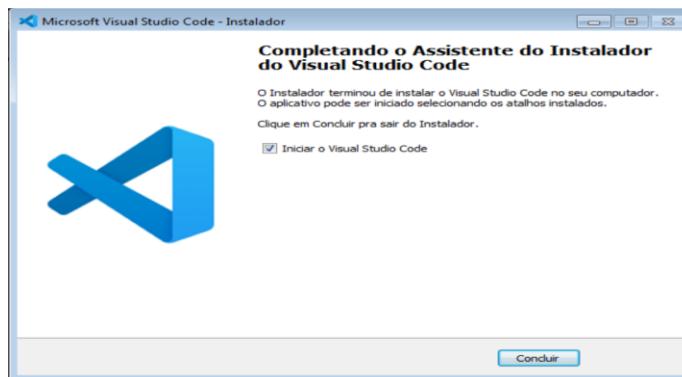
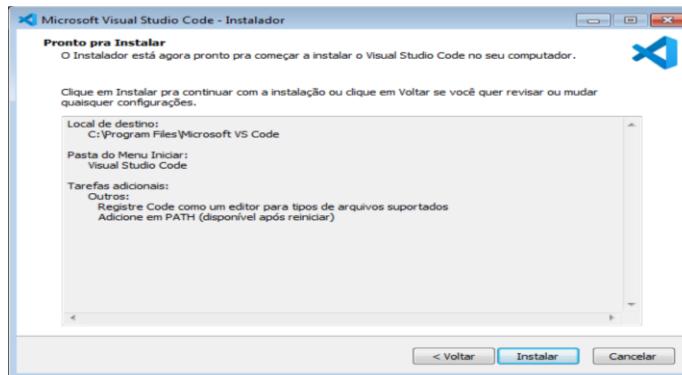
Passo 2: Instalação do executável



Compilador C para Windows

Download e instalação do VSCode

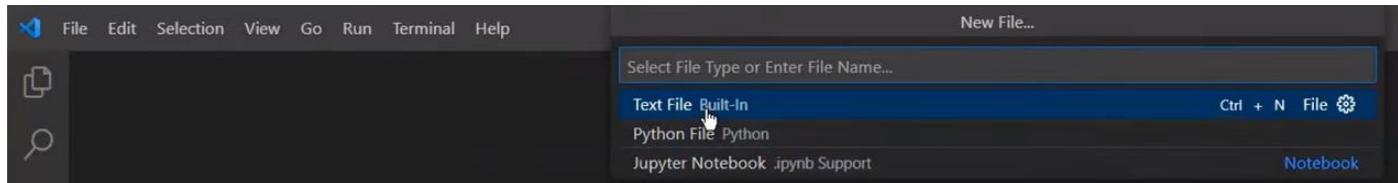
Passo 2: Instalação do executável



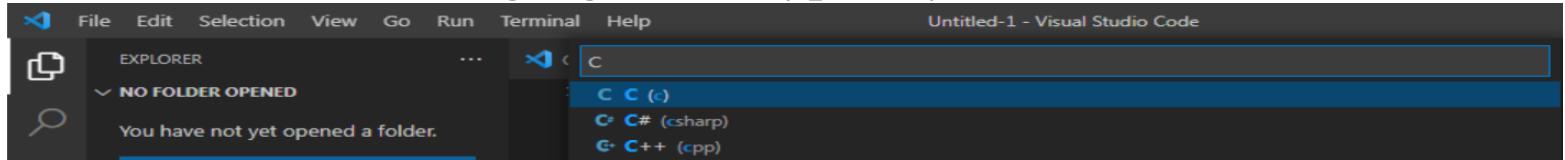
Compilador C para Windows

Download e instalação do VSCode

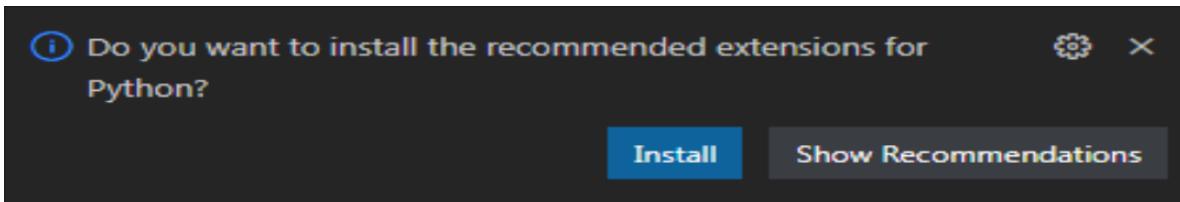
Passo 1: "File">>"New File"



Passo 2: "Select a Language" and type "Python"



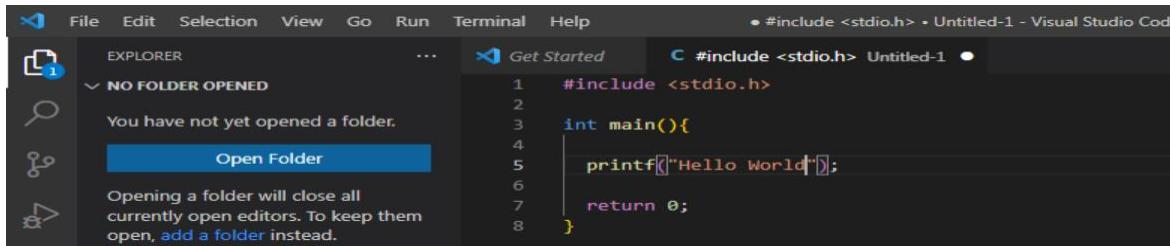
Passo 3: Caso as extensões não existam: "Install"



Compilador C para Windows

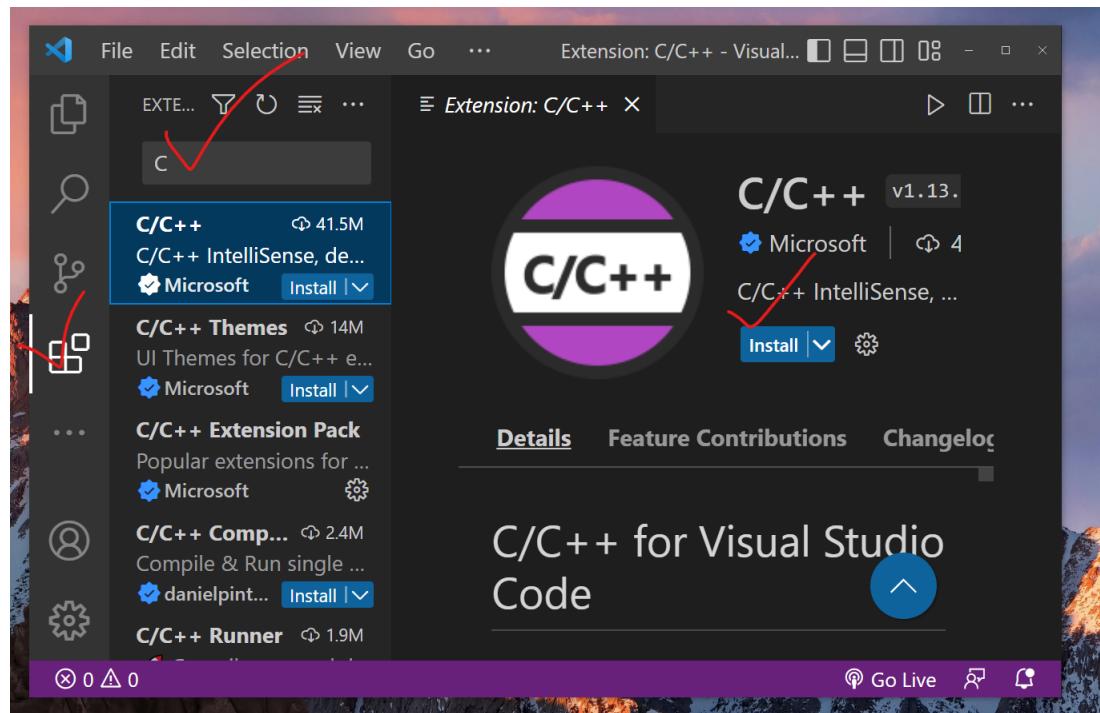
Download e instalação do VSCode

Passo 4: VS CODE pronto para o código Python



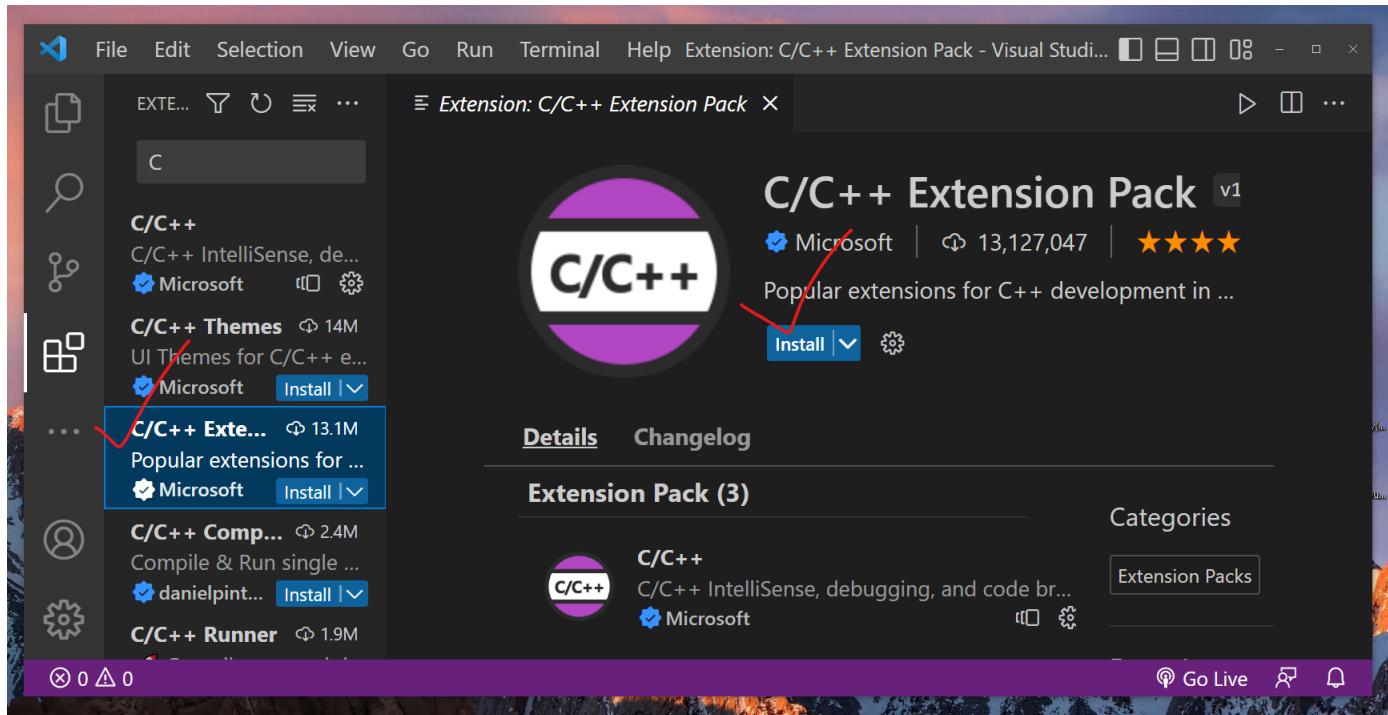
Compilador C para Windows

Instalar as extensões para C/C++



Compilador C para Windows

Instalar as extensões para C/C++



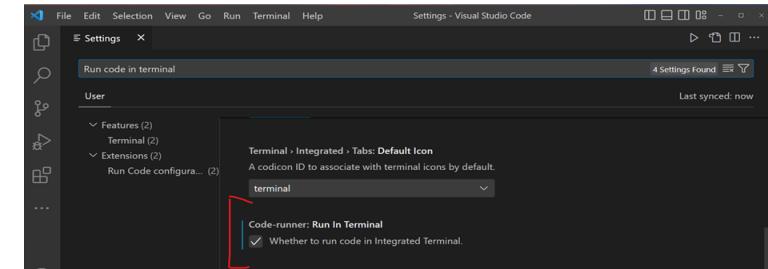
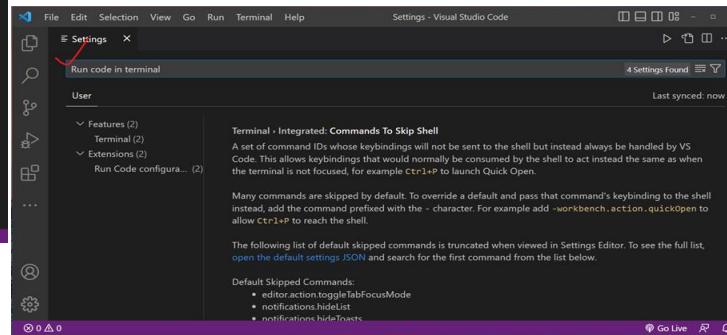
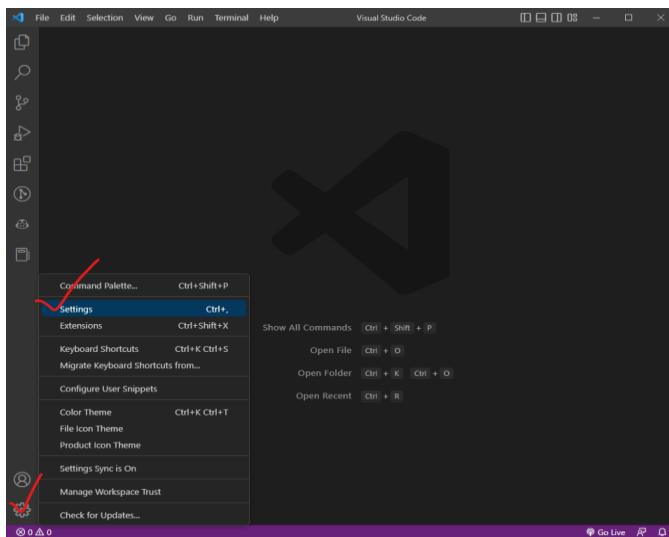
Compilador C para Windows

Instalar as extensões para C/C++



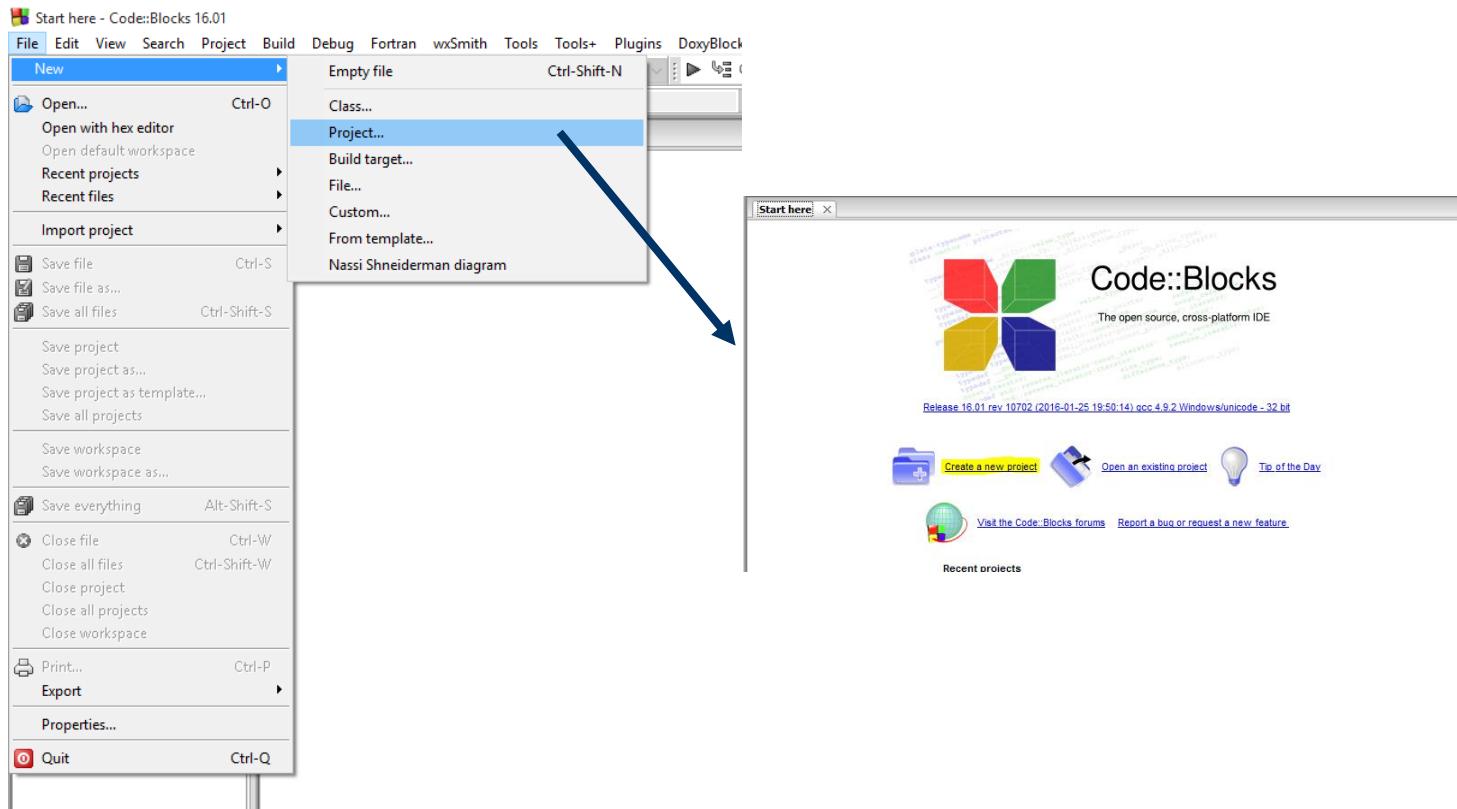
Compilador C para Windows

Instalar as extensões para C/C++



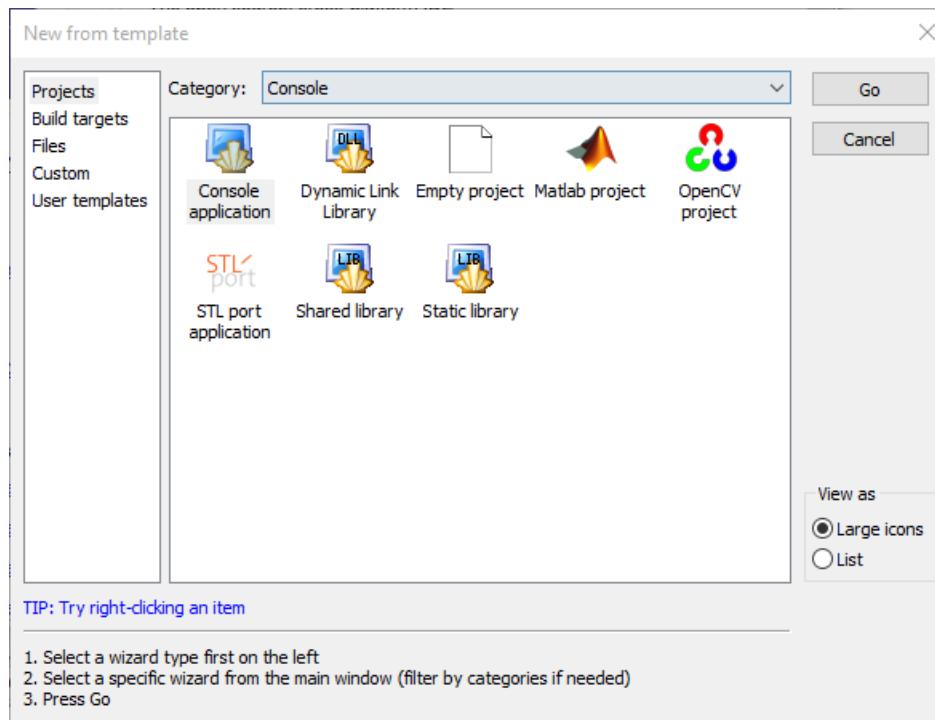
CodeBlocks

◆ 1. Criação de um projeto



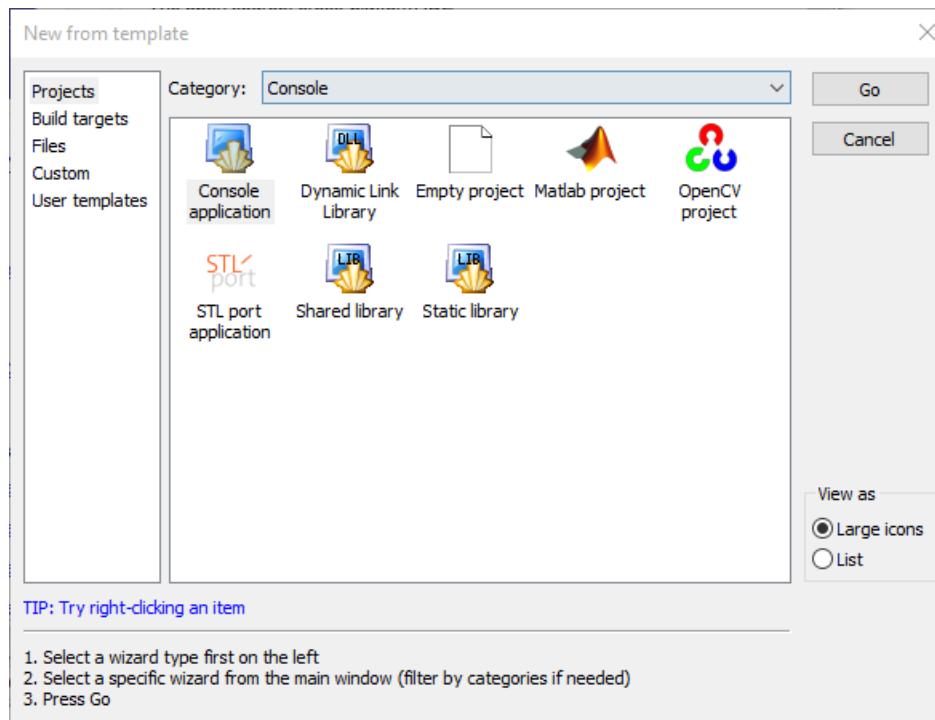
CodeBlocks

- ◆ 2. Escolha da categoria *Console*



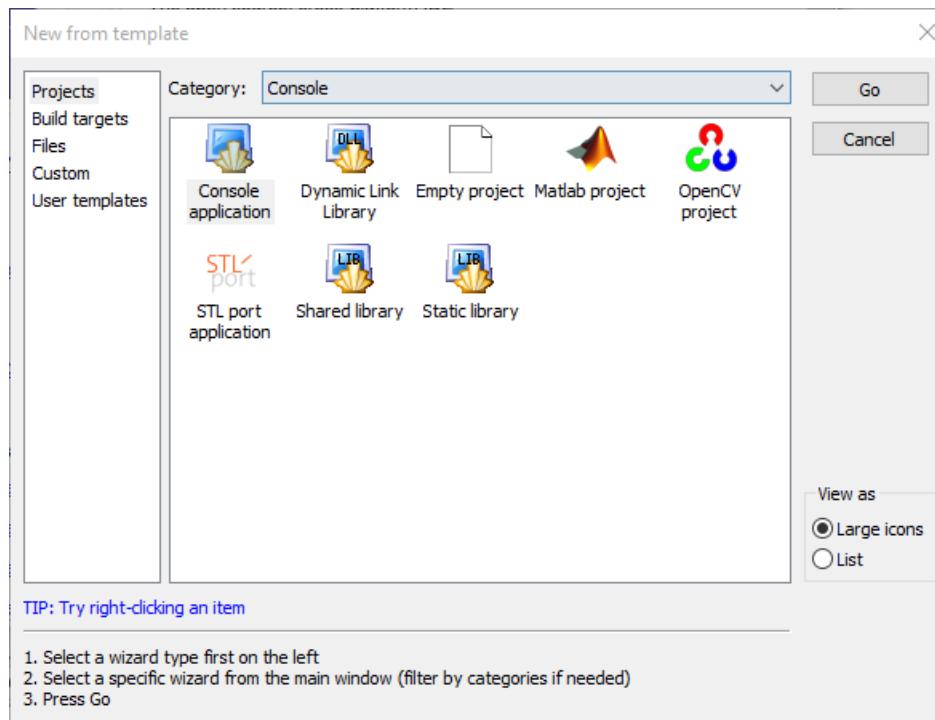
CodeBlocks

- ◆ 2. Escolha da categoria *Console*



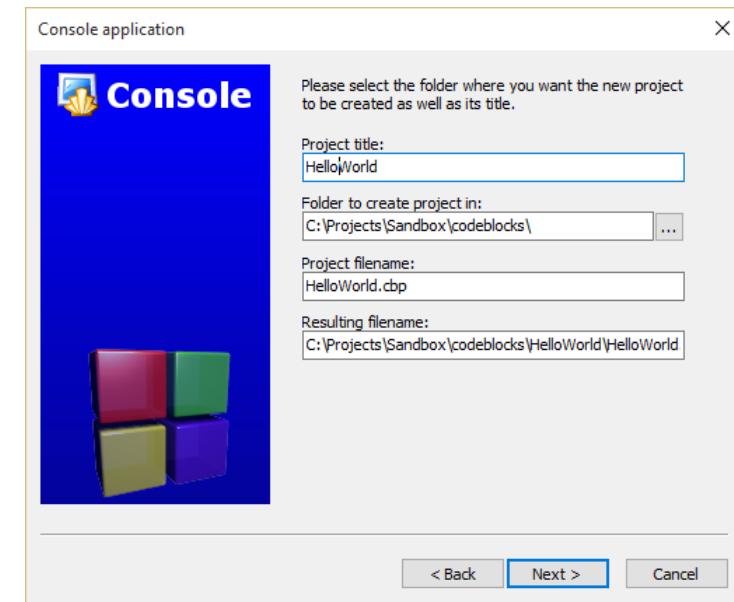
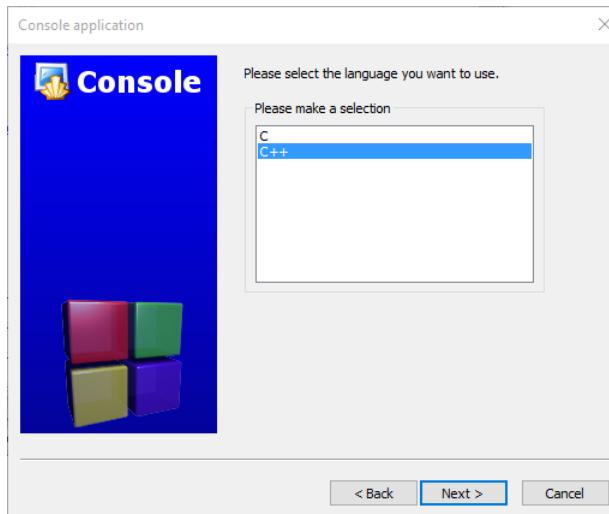
CodeBlocks

- ◆ 2. Escolha da categoria *Console*



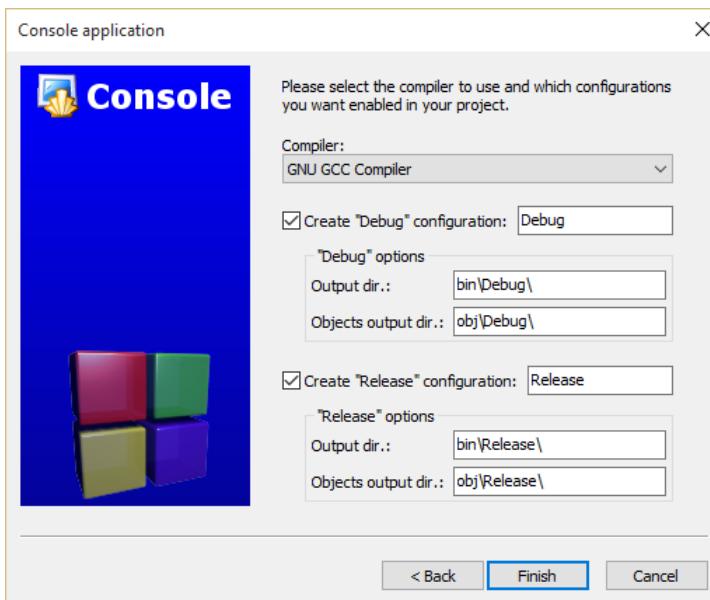
CodeBlocks

- ◆ 3. Seleção da linguagem e nome do projeto



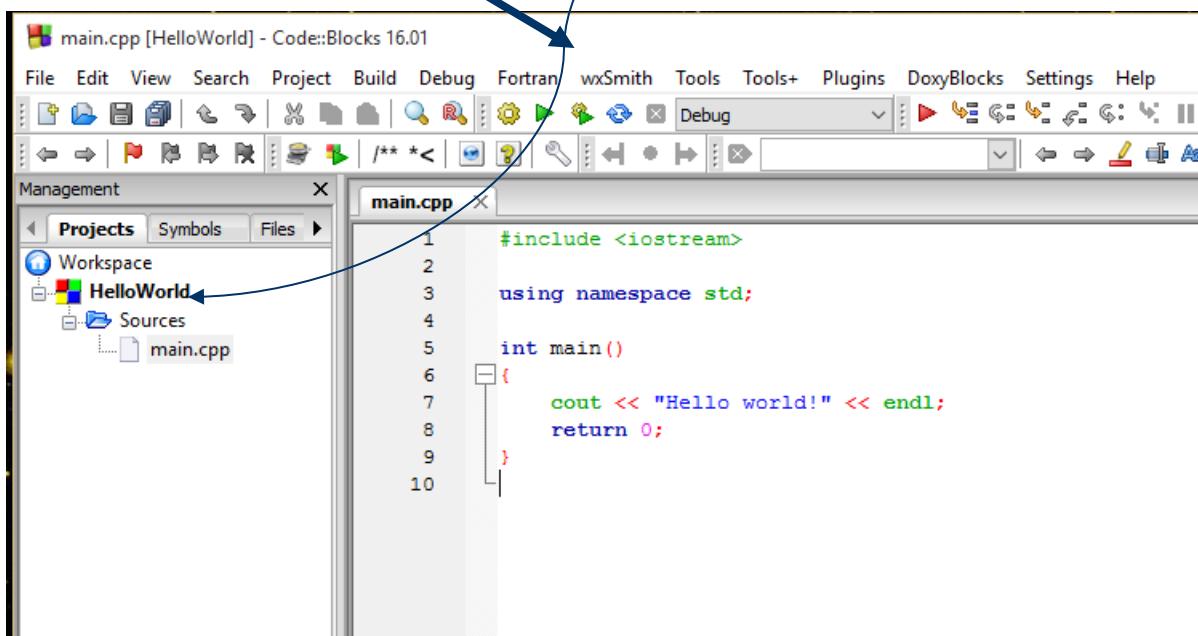
CodeBlocks

- ◆ 4. Definição do compilador



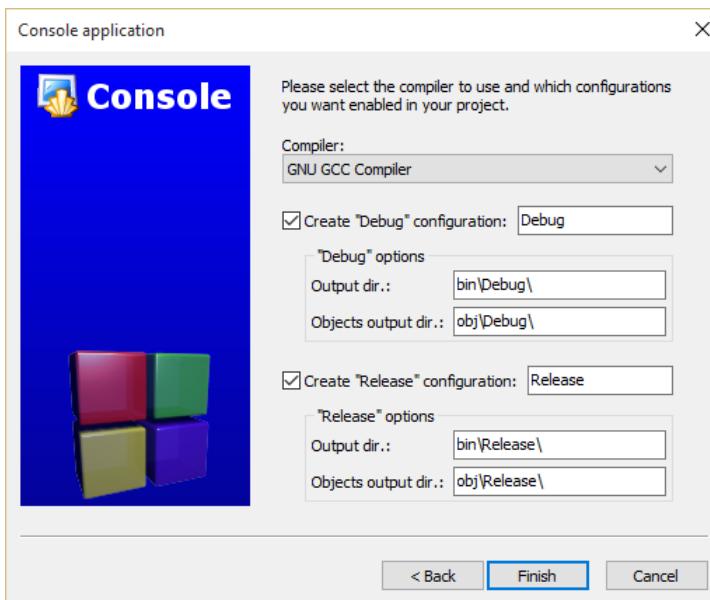
CodeBlocks

- ◆ 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:

The screenshot shows the CodeBlocks IDE interface. At the top, a window displays the console output of a 'Hello World' program. The output reads:

```
C:\Projects\Sandbox\codeblocks\HelloWorld\bin\Debug\HelloWorld.exe
Hello world!
Process returned 0 (0x0)   execution time : 0.307 s
Press any key to continue.
```

Below the console window is the 'Logs & others' dock, which contains several tabs: Code::Blocks, Search results, Cccc, Build log, Build messages, CppCheck, and CppCheck mes. The 'Build log' tab is currently selected, showing the command-line output of the compilation process:

```
----- Build: Debug in HelloWorld (compiler: GNU GCC Compiler) -----
x86_64-w64-mingw32-g++.exe -Wall -fexceptions -g -c C:\Projects\Sandbox\codeblocks\HelloWorld\main.cpp -o obj\Debug\main.o
x86_64-w64-mingw32-g++.exe -o bin\Debug\HelloWorld.exe obj\Debug\main.o
Output file is bin\Debug\HelloWorld.exe with size 2.59 MB
Process terminated with status 0 (0 minute(s), 2 second(s))
0 error(s), 0 warning(s) (0 minute(s), 2 second(s))

----- Run: Debug in HelloWorld (compiler: GNU GCC Compiler) -----
Checking for existence: C:\Projects\Sandbox\codeblocks\HelloWorld\bin\Debug\HelloWorld.exe
Executing: "C:\Program Files (x86)\CodeBlocks\cb_console_runner.exe" "C:\Projects\Sandbox\codeblocks\HelloWorld\bin\Debug\HelloWorld.exe" (in C:\Projects\Sandbox\codeblocks\HelloWorld\.)
```

CodeBlocks

- ◆ 8. Definição do compilador

- ◆ Console:

- ◆ Mensagens de compilação:

The screenshot shows the CodeBlocks IDE interface. At the top, a window displays the console output of a 'Hello World' program. The output reads:

```
C:\Projects\Sandbox\codeblocks\HelloWorld\bin\Debug\HelloWorld.exe
Hello world!
Process returned 0 (0x0)   execution time : 0.307 s
Press any key to continue.
```

Below the console window is the 'Logs & others' dock, which contains several tabs: Code::Blocks, Search results, Cccc, Build log, Build messages, CppCheck, and CppCheck mes. The 'Build log' tab is currently selected, showing the command-line build process for the 'HelloWorld' project:

```
----- Build: Debug in HelloWorld (compiler: GNU GCC Compiler) -----
x86_64-w64-mingw32-g++.exe -Wall -fexceptions -g -c C:\Projects\Sandbox\codeblocks\HelloWorld\main.cpp -o obj\Debug\main.o
x86_64-w64-mingw32-g++.exe -o bin\Debug\HelloWorld.exe obj\Debug\main.o
Output file is bin\Debug\HelloWorld.exe with size 2.59 MB
Process terminated with status 0 (0 minute(s), 2 second(s))
0 error(s), 0 warning(s) (0 minute(s), 2 second(s))

----- Run: Debug in HelloWorld (compiler: GNU GCC Compiler) -----
Checking for existence: C:\Projects\Sandbox\codeblocks\HelloWorld\bin\Debug\HelloWorld.exe
Executing: "C:\Program Files (x86)\CodeBlocks\cb_console_runner.exe" "C:\Projects\Sandbox\codeblocks\HelloWorld\bin\Debug\HelloWorld.exe" (in C:\Projects\Sandbox\codeblocks\HelloWorld\.)
```

CodeBlocks

◆ 9. Debugging



The screenshot shows the CodeBlocks IDE interface. The top menu bar includes: Fortran, wxSmith, Tools, Tools+, Plugins, DoxyBlocks, Settings, and Help. Below the menu is a toolbar with various icons for file operations, search, and navigation. The main window displays three tabs in the title bar: e.h, main.cpp, and src\rectangle.cpp. The code editor contains the following C++ code:

```
#include <iostream>

using namespace std;

#include <iostream>
using namespace std;

float RectangleArea1(float L, float W)
{
    return L*W ;
}

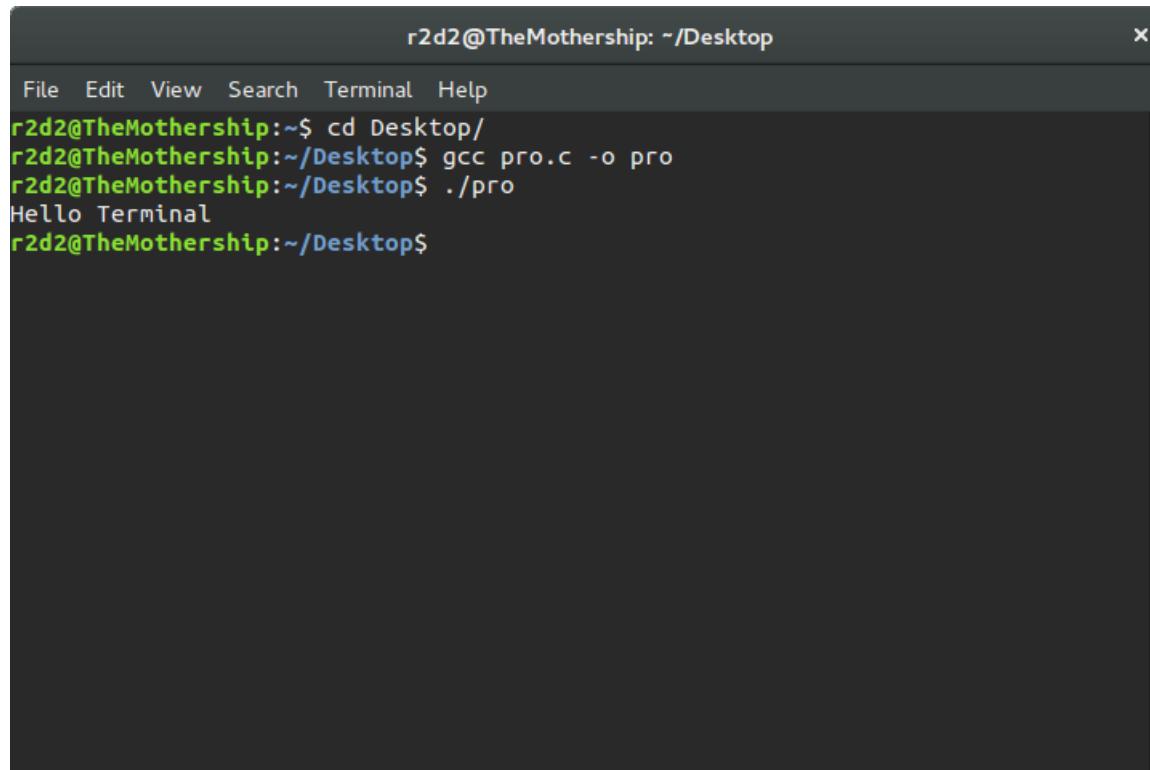
float RectangleArea2(const float L, const float W)
{
    // L=2.0 ;
    return L*W ;
}

float RectangleArea3(const float& L, const float& W)
{
    return L*W ;
}

void RectangleArea4(const float& L, const float& W, float& a
{
    area= L*W ;
}
```

Line 8 has a red dot under the opening brace, indicating a potential issue or breakpoint. A green vertical line highlights the first few lines of the code.

Compilador *gcc*



A screenshot of a terminal window titled "r2d2@TheMothership: ~/Desktop". The window has a dark theme with a light gray header bar. The terminal menu bar includes "File", "Edit", "View", "Search", "Terminal", and "Help". The command history shows:

```
r2d2@TheMothership:~$ cd Desktop/  
r2d2@TheMothership:~/Desktop$ gcc pro.c -o pro  
r2d2@TheMothership:~/Desktop$ ./pro  
Hello Terminal  
r2d2@TheMothership:~/Desktop$
```

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 terminar 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”.

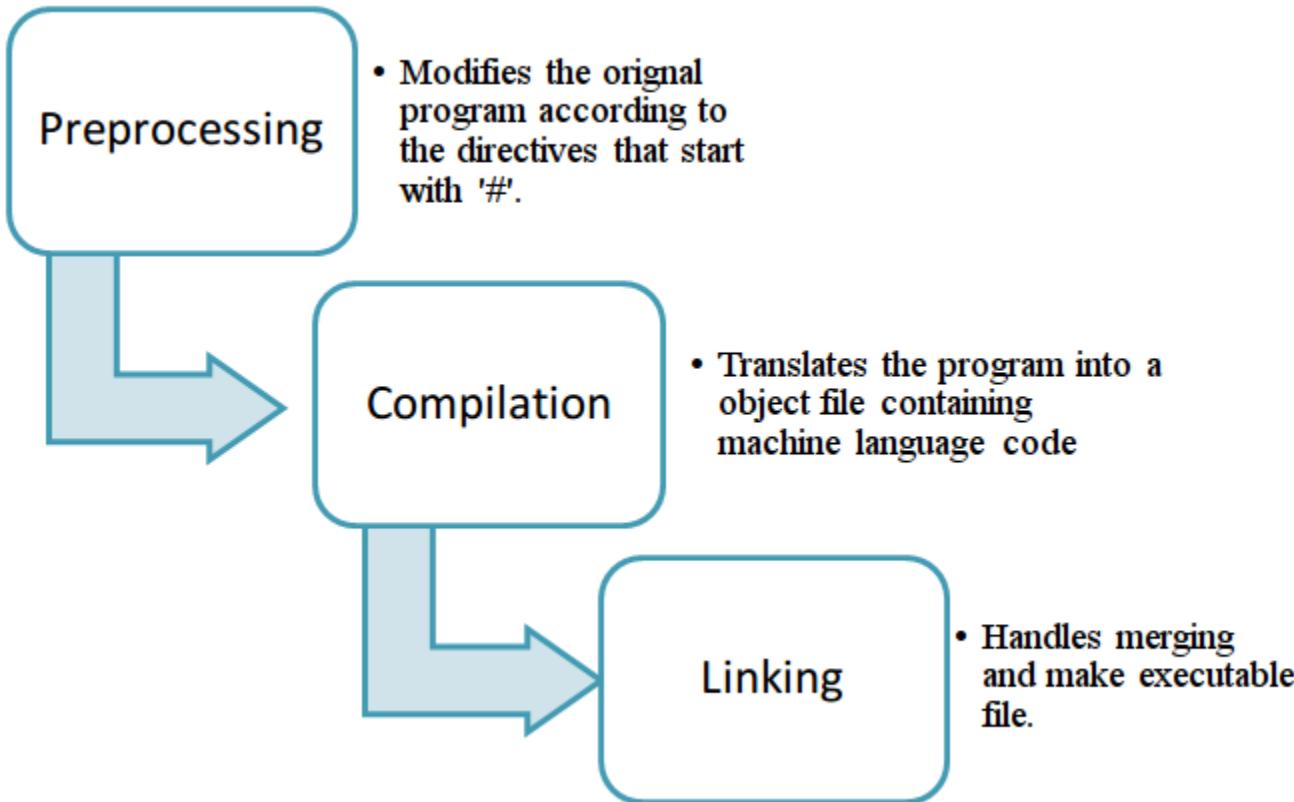
Linha 6: }

- ◆ A chaveta indica o fim do programa.

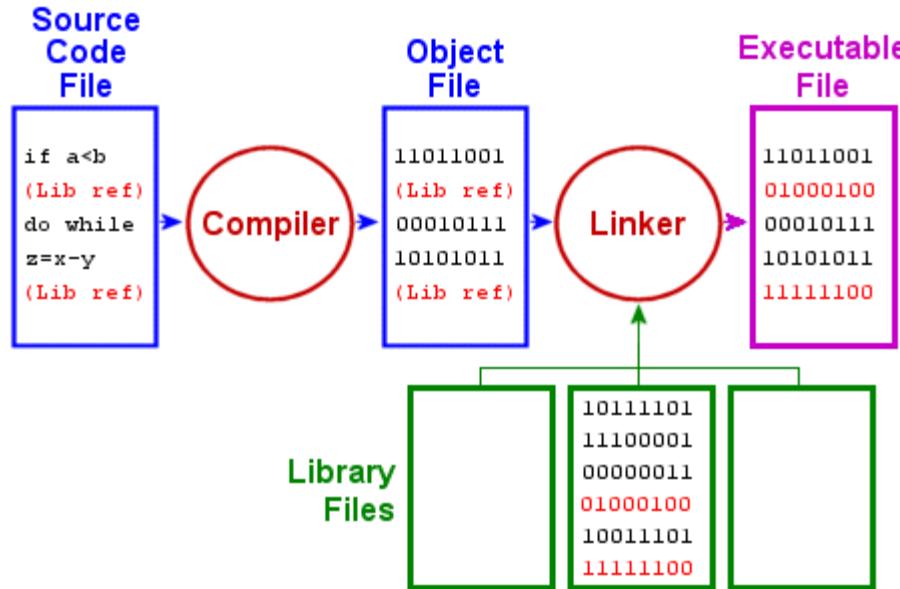
Síntaxe C

- ◆ 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

```
root@Redswitches:~#  
root@Redswitches:~# export PATH="/Dir1:$PATH"  
root@Redswitches:~# echo $PATH  
/Dir1:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/usr/games:/usr/local/games:/snap/bin  
root@Redswitches:~#
```

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>

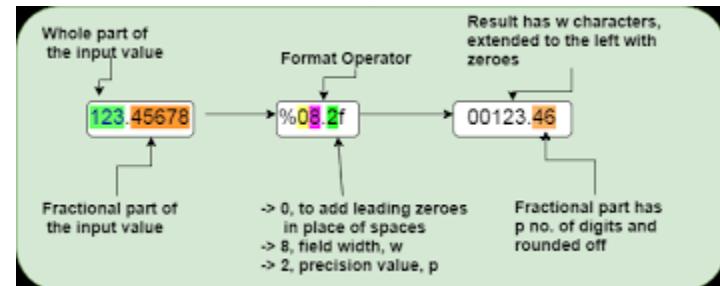
int main()
{
    printf("Hello everybody, my name is....!\n");
    return (1);
}
```

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
\'	
\1	Linefeed character
\0	Null 'character'

Outros detalhes como a precisão podem ser especificados, com a seguinte sintaxe:

[flags][width].[precision]



printf() –Especificadores de formato

Outros detalhes como a precisão podem ser especificados, com a seguinte sintaxe:

[flags][width].[precision]

printf("number=%3d\n", 10);

→ number =

	1	0
--	---	---

printf("number=%2d\n", 10);

→ number =

1	0
---	---

printf("number=%1d\n", 10);

→ number =

1	0
---	---

printf("number=%7.2f\n", 5.4321);

→ number =

			5	.	4	3
--	--	--	---	---	---	---

printf("number=%.2f\n", 5.4391);

→ number =

5	.	4	4
---	---	---	---

printf("number=%f\n", 5.4321);

→ number =

5	.	4	3	2	1	0	0
---	---	---	---	---	---	---	---

printf() –Especificadores de formato

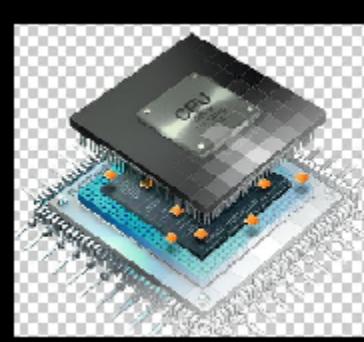
```
printf("a = %d\nb = %d\n", a, b);
```

ControlString

1st Line 2nd Line arg1 arg2

New Line New Line

```
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*

```
printf("%d %c %d = %d", a, 0x58, b, c)
```

A diagram illustrating the mapping of printf format specifiers to their corresponding arguments. The text "printf("%d %c %d = %d", a, 0x58, b, c)" is at the top. Below it, four colored arrows point from specific parts of the format string to the arguments: a green arrow points from "%d" to "a"; a blue arrow points from "%c" to "0x58"; a magenta arrow points from "%d" to "b"; and a cyan arrow points from "= %d" to "c".

Tipos de dados básicos

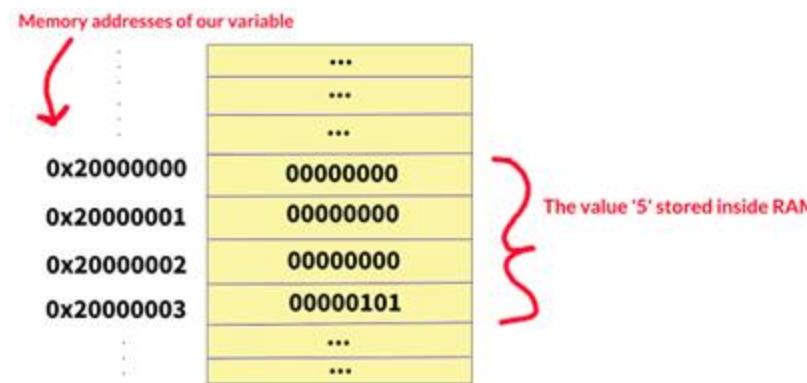
- ◆ int – 2 byte integer
- ◆ long – 4 byte integer
- ◆ float – 2 byte floating point (real)
- ◆ double – 4 byte floating point
- ◆ char – single byte character
- ◆ unsigned char – single unsigned byte

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

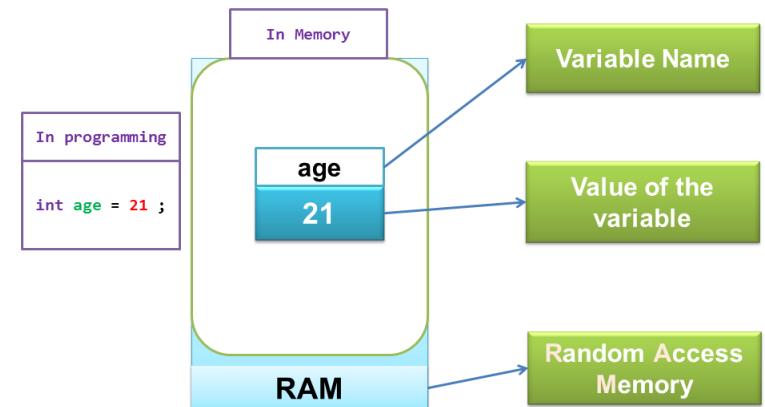
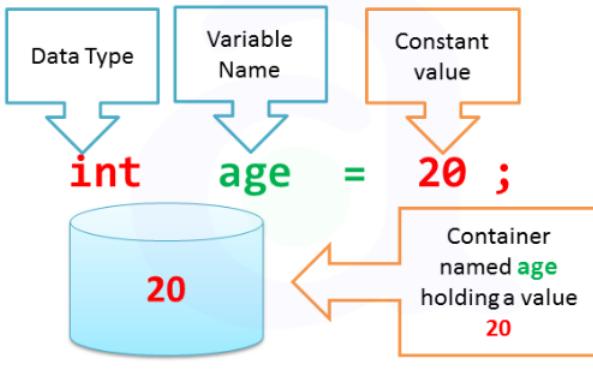
Variáveis

- ◆ 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:**

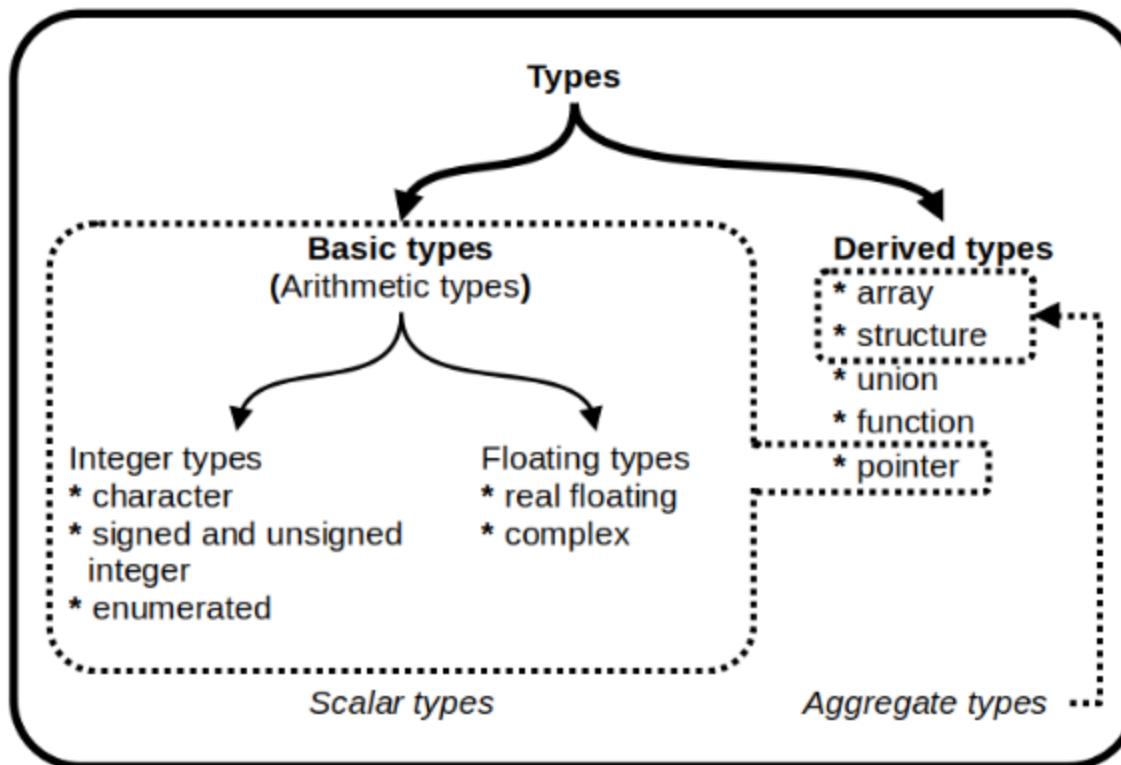
- Só pode conter letras, dígitos ou _
- O primeiro caractere deve ser uma letra ou _
- As palavras reservadas do C, não podem ser usadas como identificadores

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
\geq	Maior ou igual a
<	Menor que
\leq	Menor ou igual a
\equiv	Igual a
\neq	Diferente de

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

Operadores de incremento/decremento

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

Incremento

`x = x + 1;`

`x++;`

Decremento

`x = x - 1;`

`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
	=	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--);
```

```
}
```

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

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

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

```
int age ;  
printf("Enter your age : ");  
scanf("%d", &age);
```

scanf reads an integer(a number)
which the user enters

scanf puts that read value
“At the address of” ‘age’ variable

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");
}
```

Terminal output

Memória

num

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:

Memória

num

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

Memória

num	56
-----	----

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



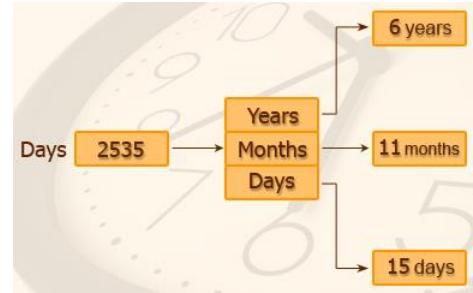
Exercícios sobre o *scanf*

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

Format string

Address of
a variable

Laboratório 3



```
#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

$$y = 2535 / 365 = 6,945\ldots = 6$$

$$ndays = 2535 - 6*365 = 345$$

$$m = 345 / 30 = 11,5 = 11$$

$$d = 345 - 11*30 = 15$$

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

```
#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 directória *default* do sistema

`#include "filename.h"` para bibliotecas na directória 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

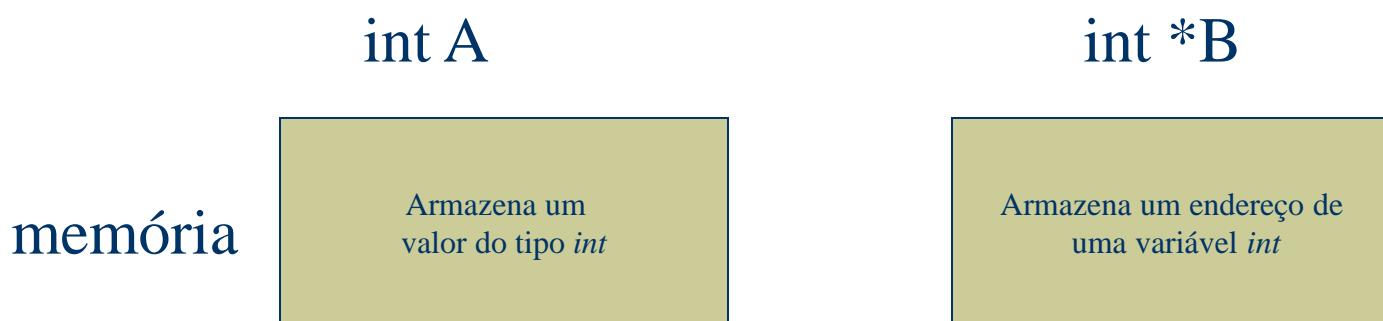
int main()
{
    printf("sum(10, 20) = %.2f\n", sum(10, 20));
    printf("sub(10, 20) = %.2f\n", sub(10, 20));
    printf("mult(10, 20) = %.2f\n", mult(10, 20));
    printf("div(10, 20) = %.2f\n", div(10, 20));
    printf("mod(10, 20) = %d\n", mod(10, 20));

    return 0;
}
```

Compilação e *linkagem*:
`gcc -o arith arith.c`

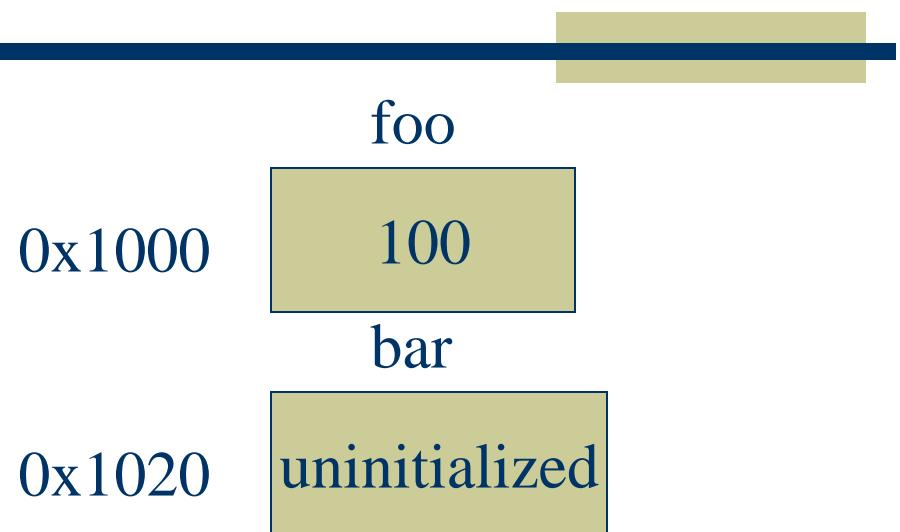
Introdução aos ponteiros

- ◆ 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

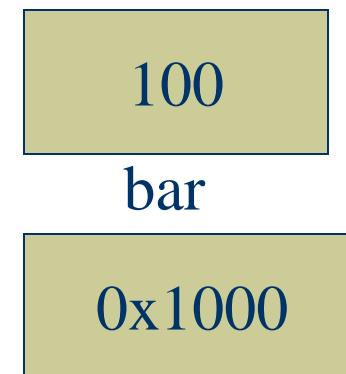
```
int foo = 100;
```

```
int *bar;
```

```
bar = &foo;
```

0x1000

0x1020



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;
```

0x1000

foo

200

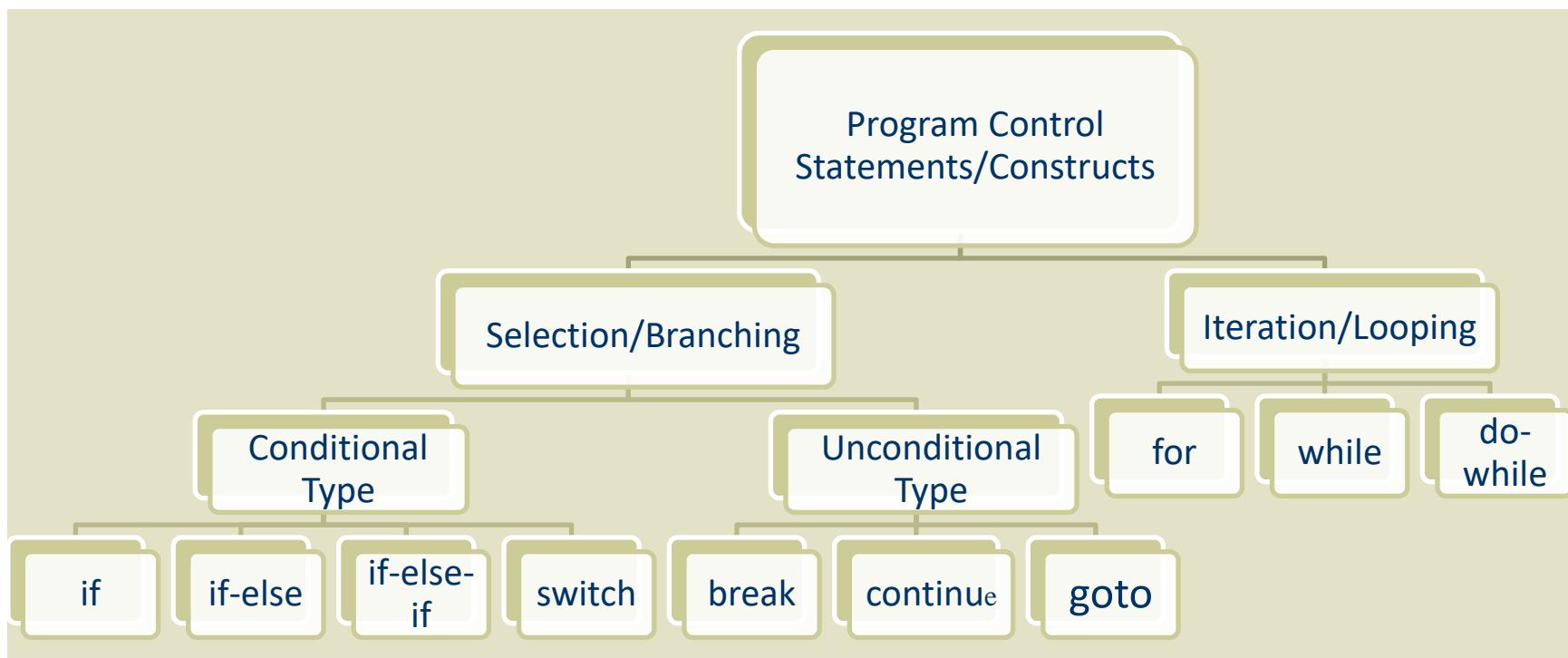
bar

0x1000

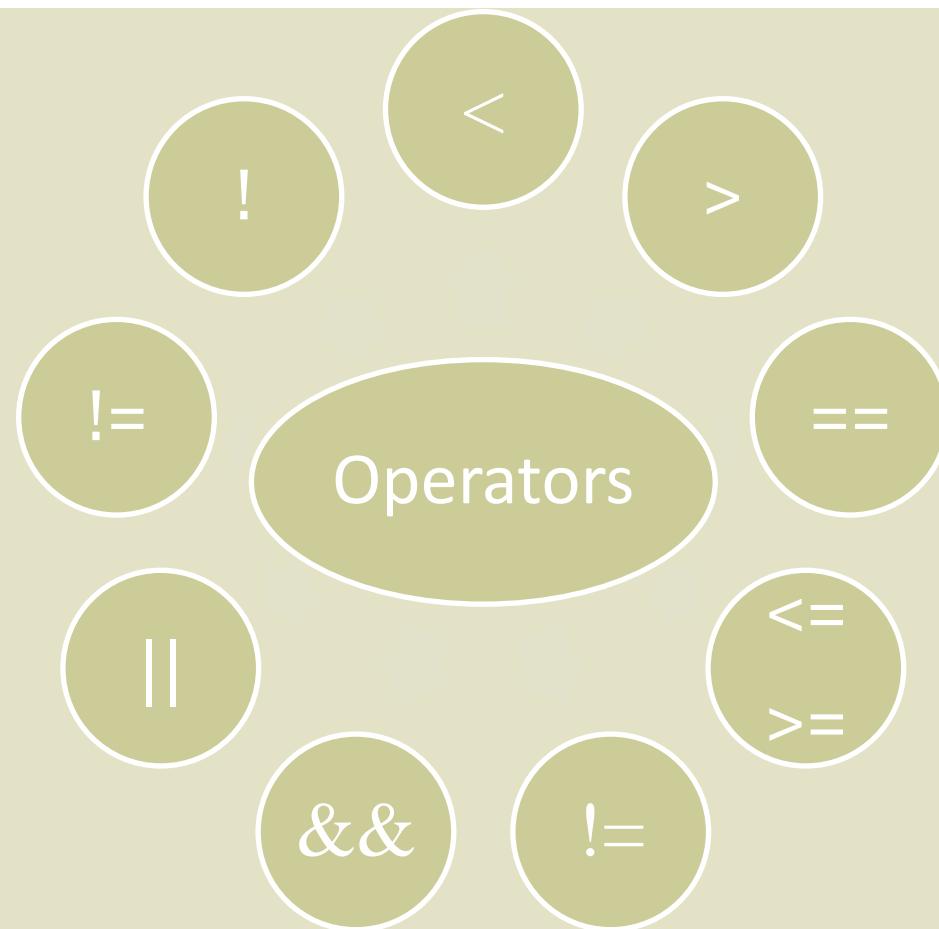
Qual a diferença ?

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

Estruturas de controlo



Estruturas de controlo



Estruturas de controlo - *if*

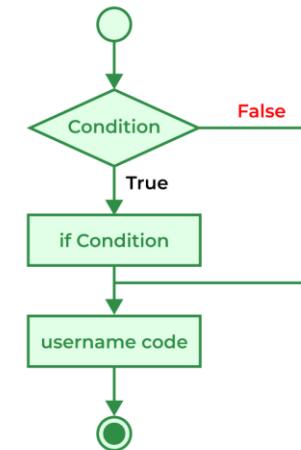
- ◆ Estrutura de controlo que executa um bloco de instruções se uma **determinada** condição é verdadeira

- ◆ Síntaxe:

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

- ◆ Exemplo:

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



Estruturas de controlo - *if*

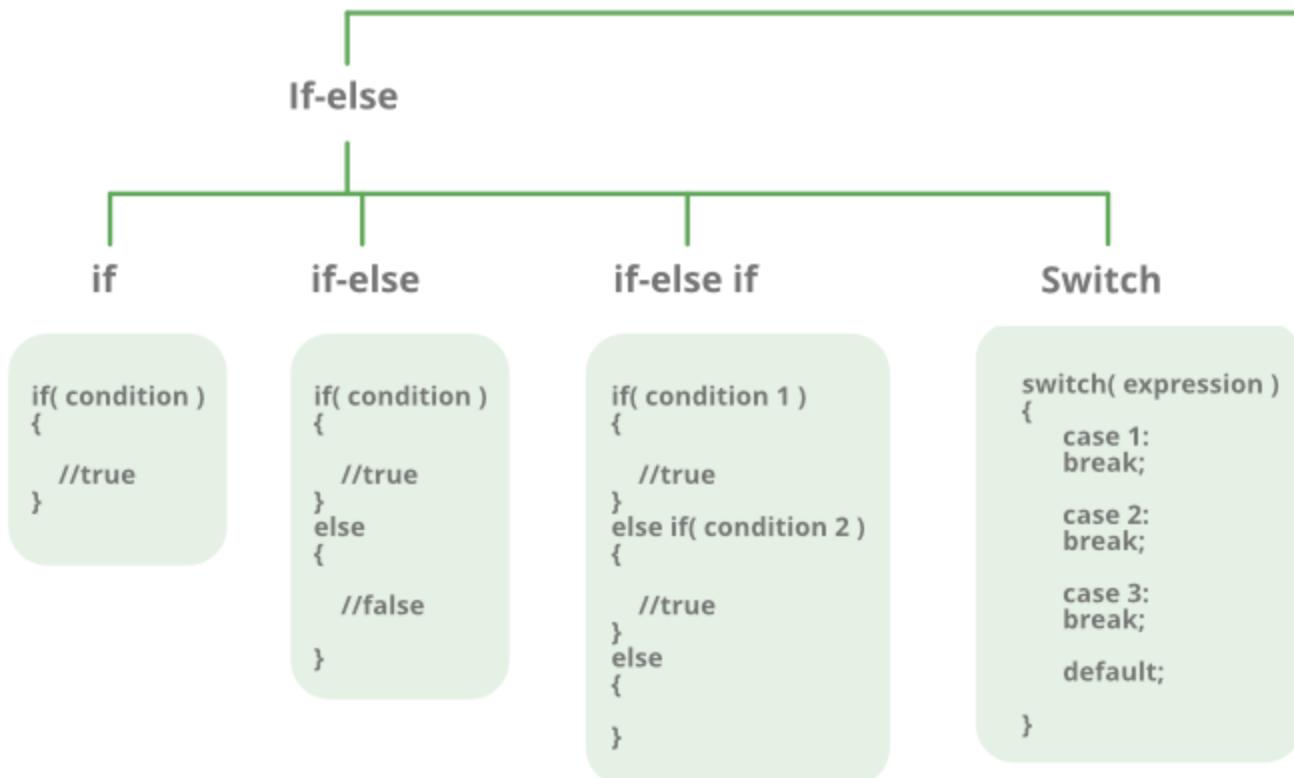


Tabela ASCII

```
#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;
}
```

```
#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 );
}
```

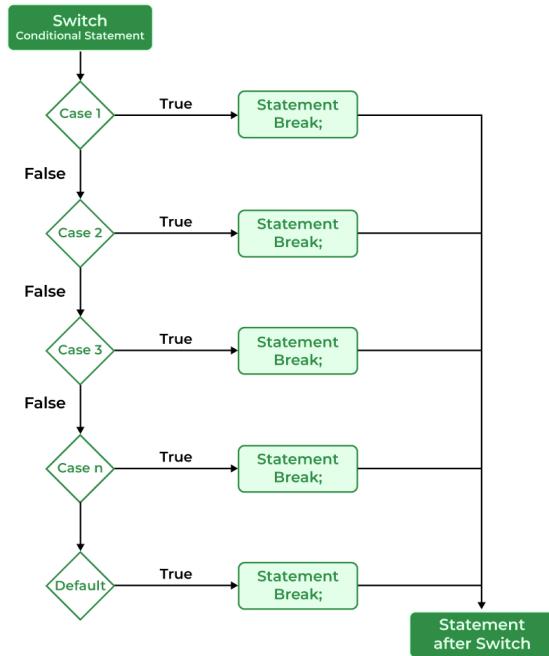
```
cook@pop-os:~$ ascii -d
      0 NUL   16 DLE   32   48 0   64 @   80 P   96 `   112 p
      1 SOH   17 DC1   33 !   49 1   65 A   81 Q   97 a   113 q
      2 STX   18 DC2   34 "   50 2   66 B   82 R   98 b   114 r
      3 ETX   19 DC3   35 #   51 3   67 C   83 S   99 c   115 s
      4 EOT   20 DC4   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 l
     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
```

```
#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

```
#include <stdio.h>
int main() {
    int num = 8; ①
    ② switch (num) {
        case 7:
            printf("Value is 7");
            break;
        case 8: ③
            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 definir um conjunto constantes
- ◆ Apresenta uma numeração para além do descritivo

```
enum week{sun, mon, tue,...};
```

↑ ↑ ↑
Keyword data type members

Enumeration

Enumerações

```
#include <stdio.h>

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

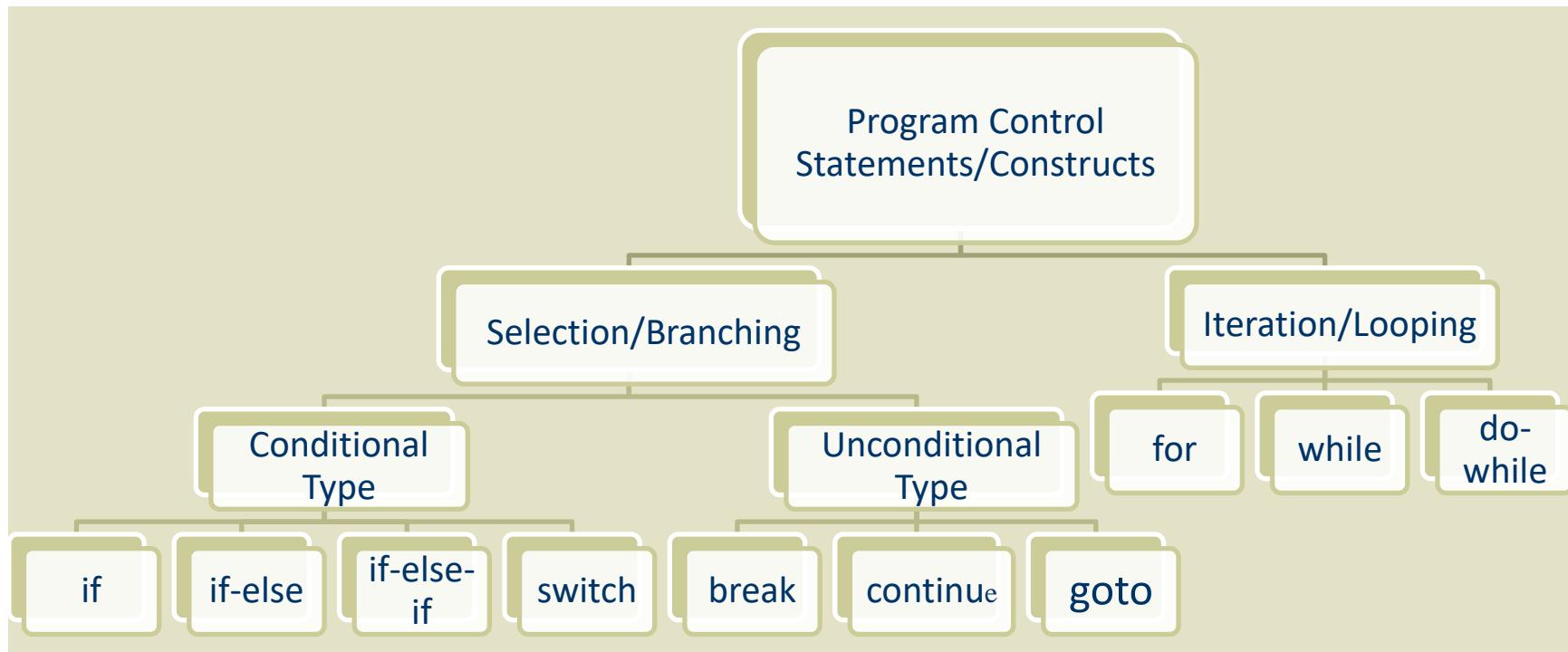
int main() {
    // Initializing enum variable
    enum textEditor feature = BOLD;
    printf("Selected feature is %d\n", feature);

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

    return 0;
}
```

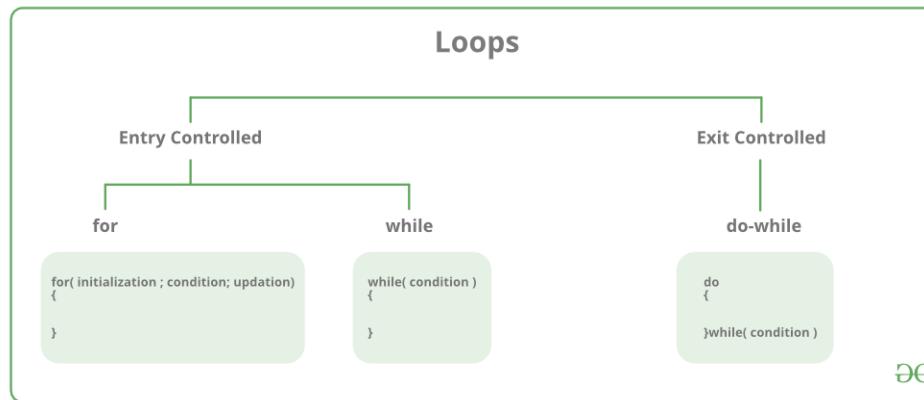
- ◆ Uma variável do tipo enumerado, pode assumir o descritivo ou o índice definido
- ◆ Só a escrita do índice pode ser efetuada
- ◆ Caso não seja definido qualquer índice é assumido o “0” no primeiro elemento
- ◆ Caso algum elemento não tenha índice, assume o índice anterior + 1

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*

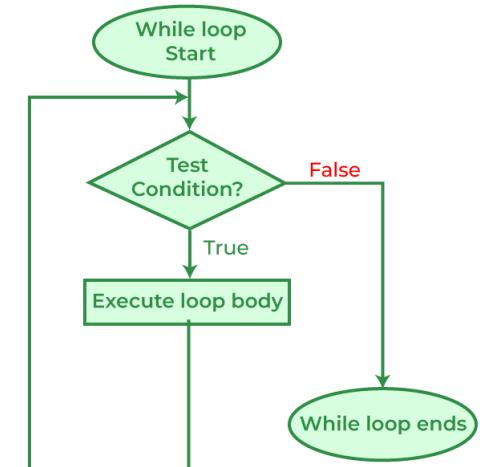
Declaring and Initializing loop control variable
for (int i =0; i<10 ; i++) {
 // Loop statements to be executed
}

```
#include <stdio.h>

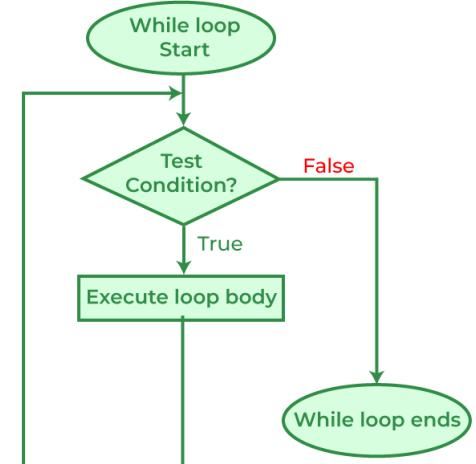
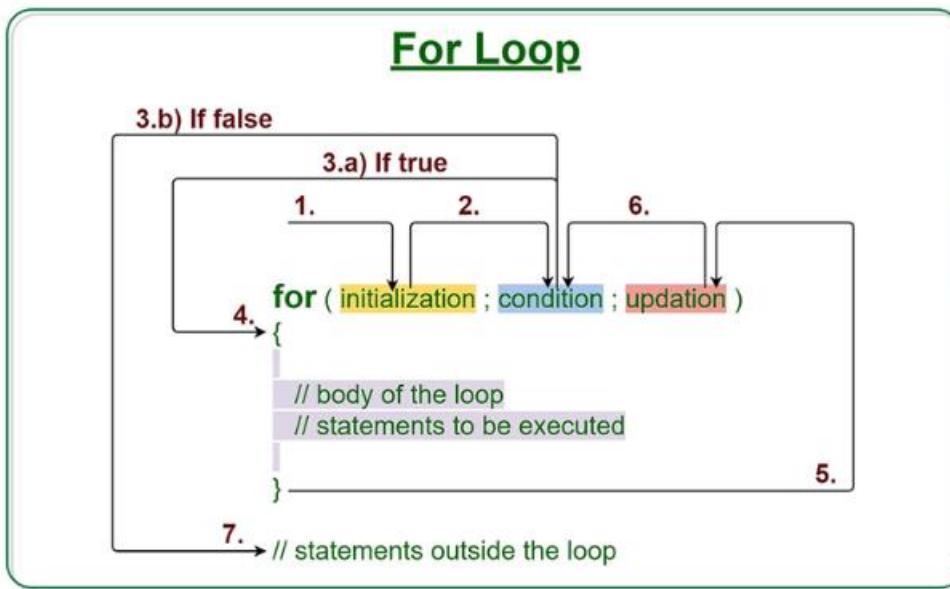
int main () {
    int a;

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

    return 0;
}
```



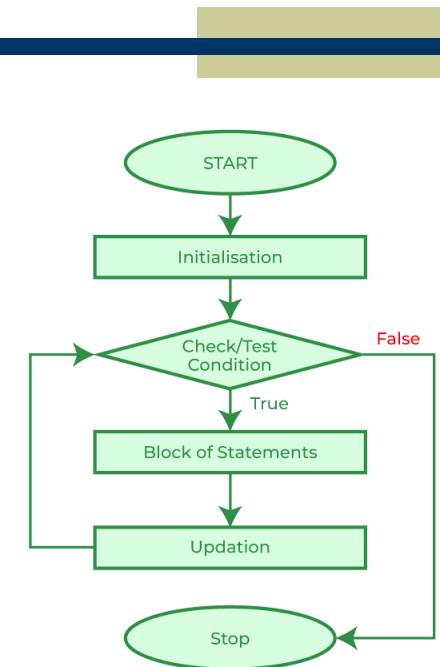
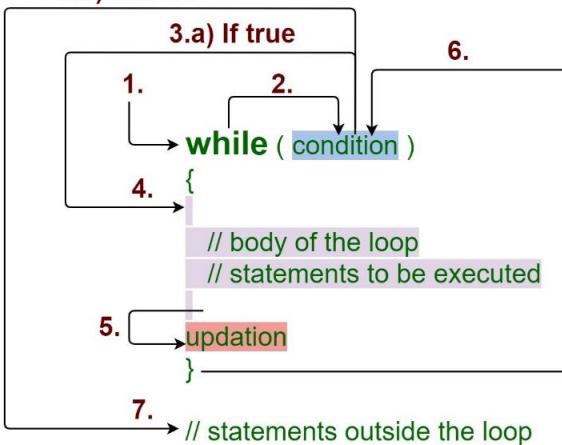
Estruturas iterativas - *for*



Estruturas iterativas – *while*

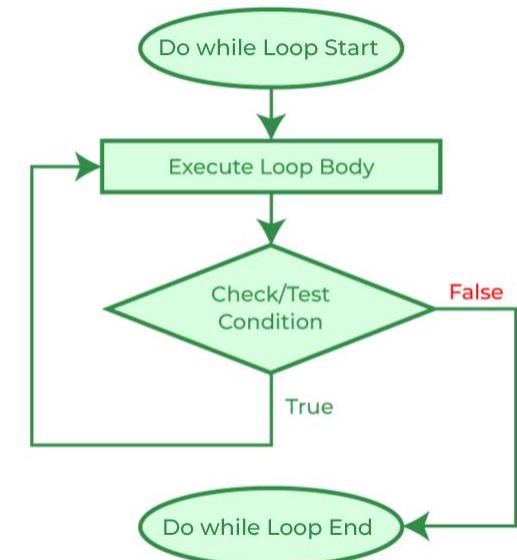
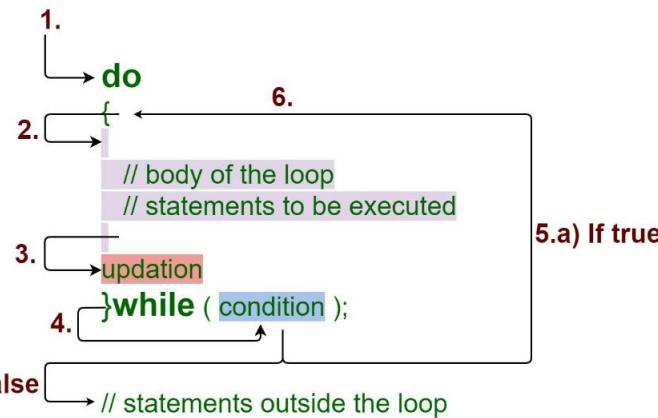
While Loop

3.b) If false



Estruturas iterativas – *do..while*

Do - While Loop



```
#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

```
#include <stdio.h>

int main()
{
    int i ;

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

    return 0;
}
```

```
#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;
}
```

```
#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

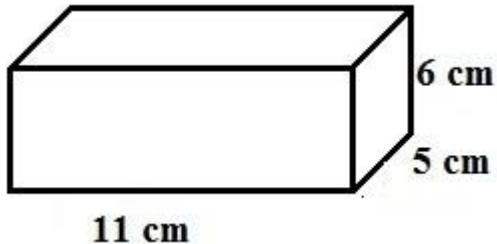
volume.c

```
#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];
```

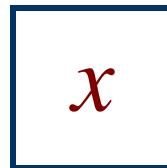
- ◆ Every char of name can be accessed as name[i]
- ◆ Arrays are indexed from 0 so the first character in a string is string[0]

Sample program

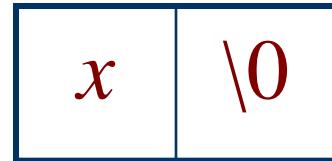
```
int main()
{
    char name[40];
    printf("what is your name? ");
    scanf("%s" name);
    printf("hello %s.\n" name);
    return 0;
}
```

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	<code>~</code>
AND	<code>&</code>
OR	<code> </code>
XOR	<code>^</code>

- ◆ bitwise shifts

shift left	<code><<</code>
shift right	<code>>></code>

- ◆ boolean logic

Not	<code>!</code>
And	<code>&&</code>
Or	<code> </code>

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);
```

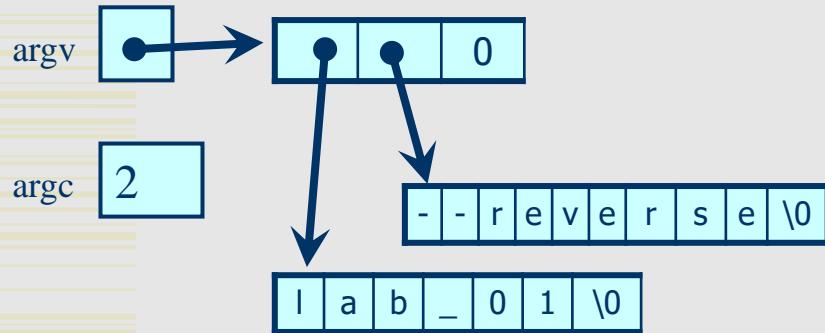
Example

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

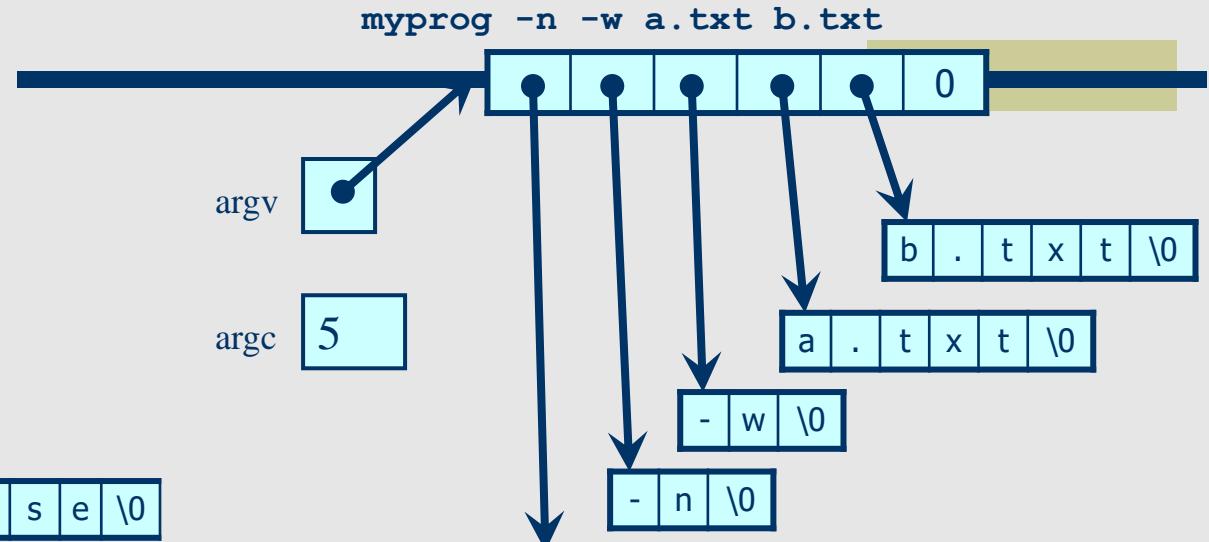
Coding: Processing cmd args

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

lab_01 --reverse



vector<string> arg(argv, argv+argc);



Alocação de memória

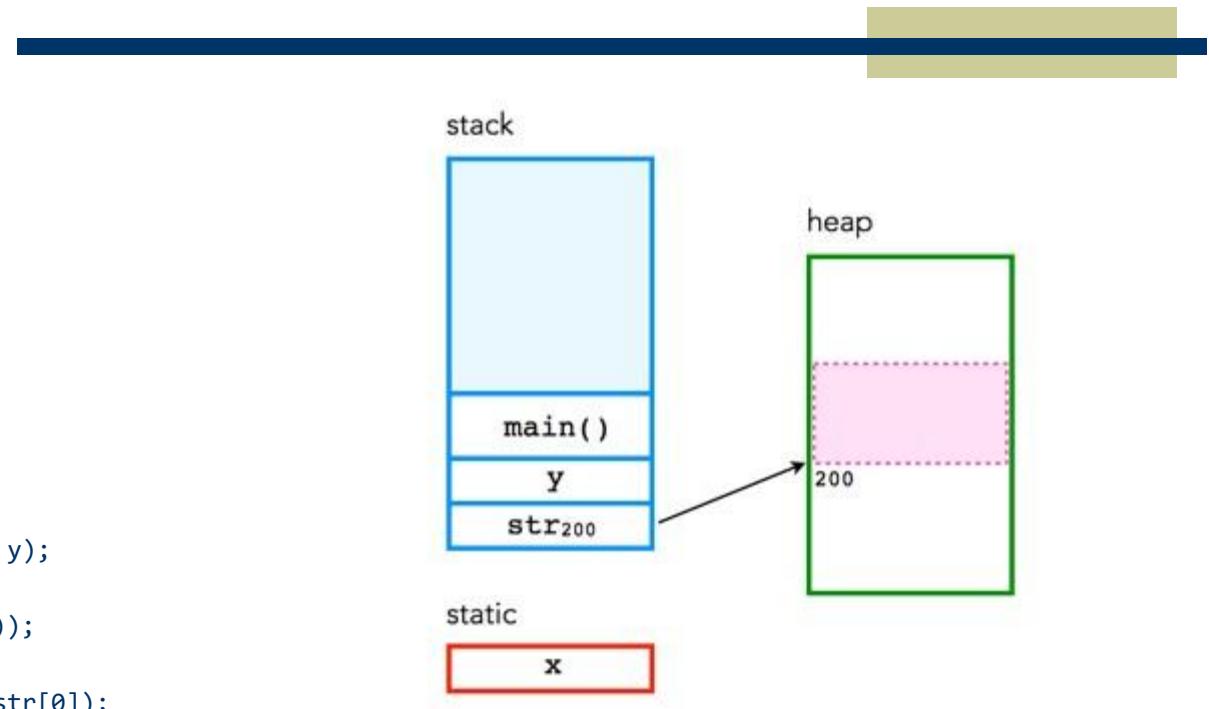
```
#include <stdio.h>
#include <stdlib.h>

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).