C++ Software Engineering

for engineers of other disciplines

Module 1

"C++ Syntax"

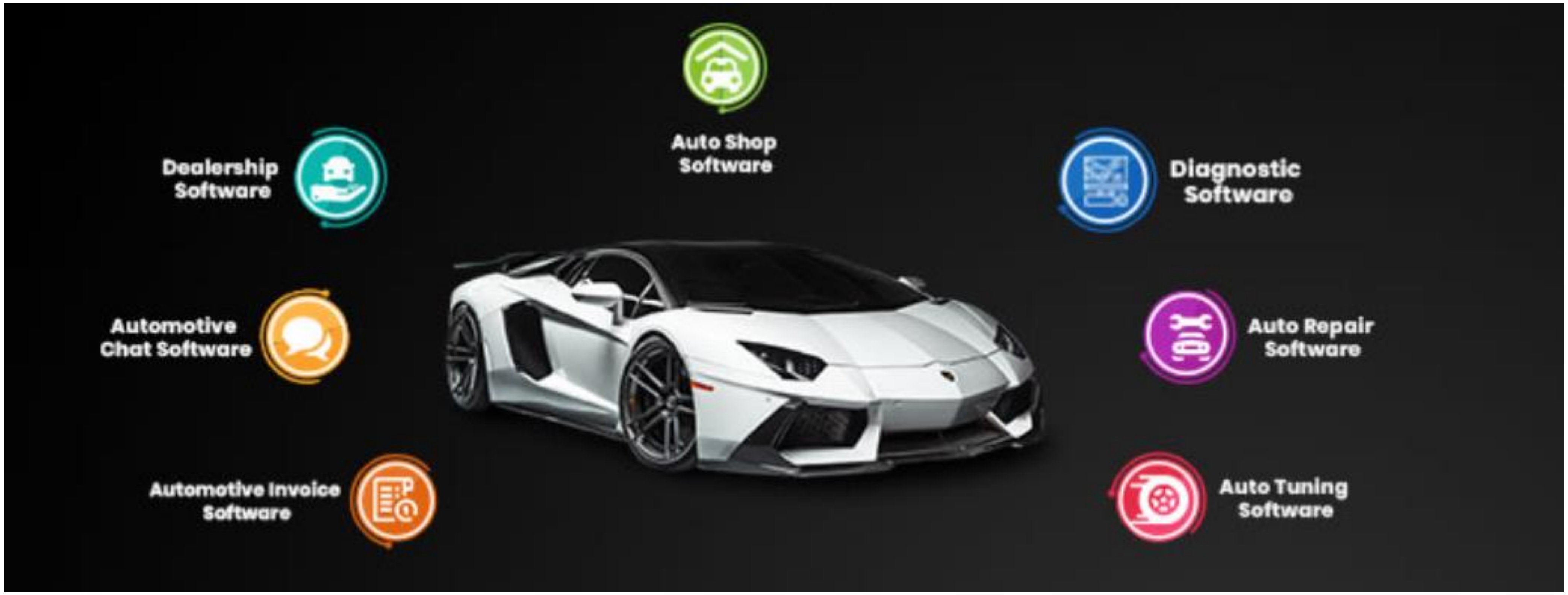
1st Lecture: Hello World!

Spring 2022
Gothenburg, Sweden

Introductions *"in 30 seconds"*

- ❑ Name
- ❑ Job
- ❑ Objectives of Enrolling Bootcamp.

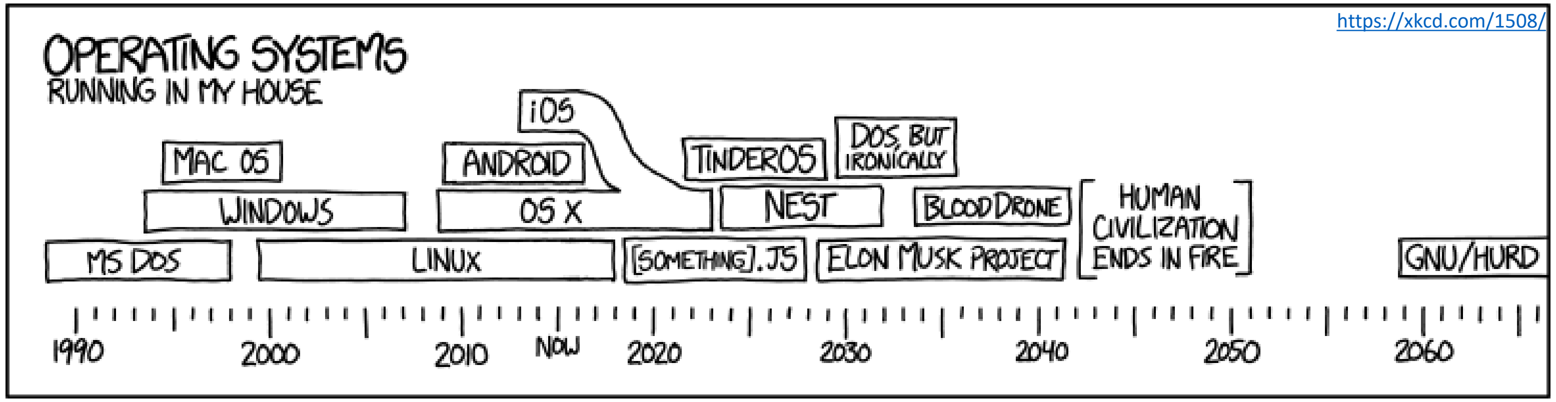
Software in the Automotive Industry



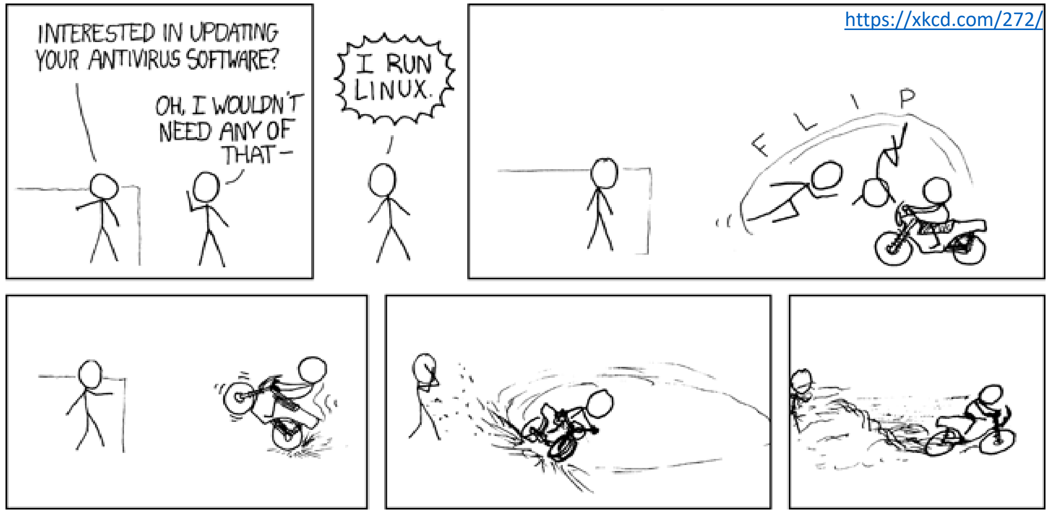
https://www.excellentwebworld.com/automotive-software-development/

Work Environment

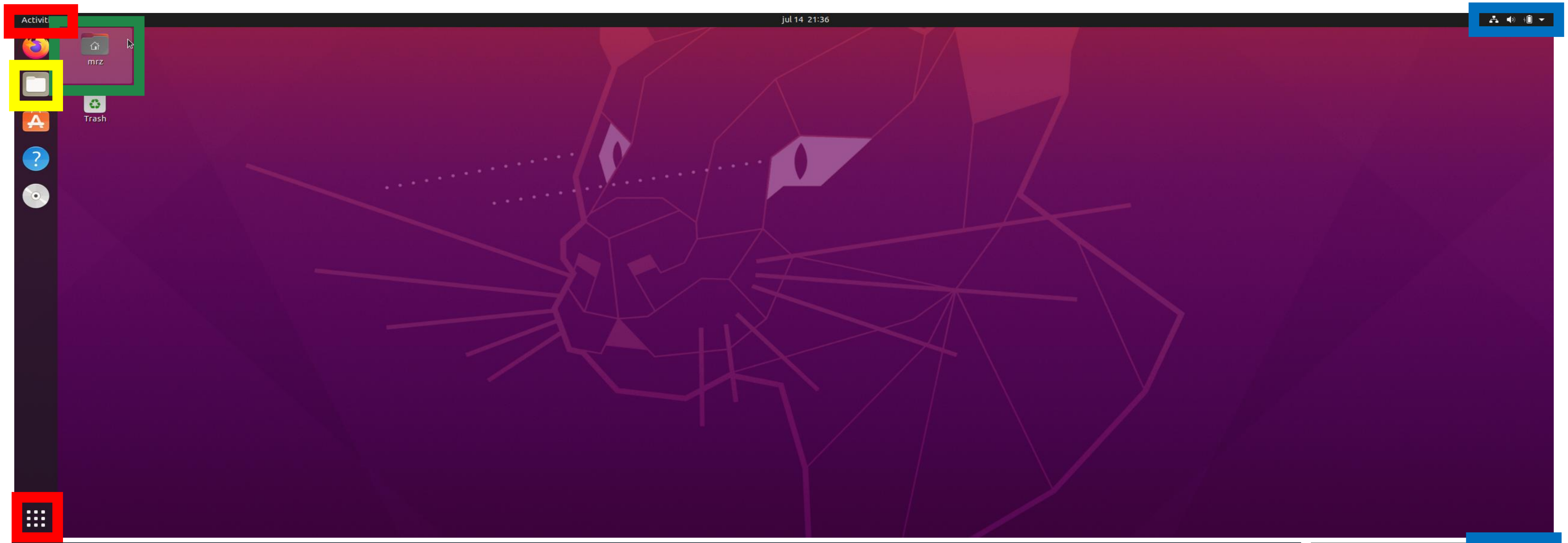
- Operating System
  - GNU
  - Linux
  - Debian
  - Ubuntu

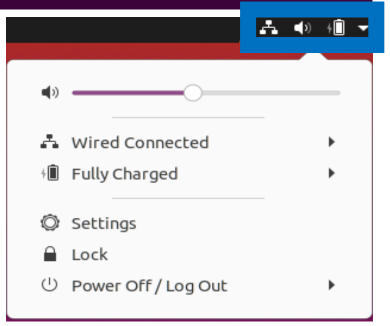


https://xkcd.com/1508/


INTERESTED IN UPDATING YOUR ANTIVIRUS SOFTWARE?
OH, I WOULDN'T NEED ANY OF THAT –
I RUN LINUX.
F L I P

https://xkcd.com/272/



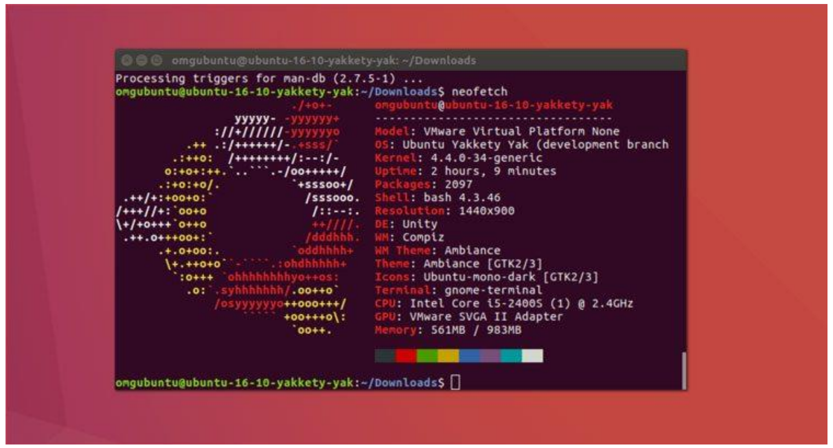


- Click on the corners highlighted by the **red boxes** in the above screenshot to search for applications, settings, and etc. The same could be achieved by pressing the *"Windows key"* on your keyboards. Press the *"Escape Key"* `[Esc]` to exit the menu.
- The **blue box** is the system tray, where you can find system controls, and network manager. Click on the tray to expand the illustrated menu.
- The **yellow box** is the file explorer, and the **green box** opens a file explorer in your *"home"* directory.

Development Environment

- Integrated Development Environment (IDE)
    - Integrates:
        - Text Edition Environment
        - Compilation Environment
        - Execution Environment
        - Debugging Environment
        - And many more Environments!
    - Boosts Productivity
    - Personal Preferences Matter!

- Terminal Emulator
    - Command-Line Interface to OS's Services
    - GNOME (bash shell)

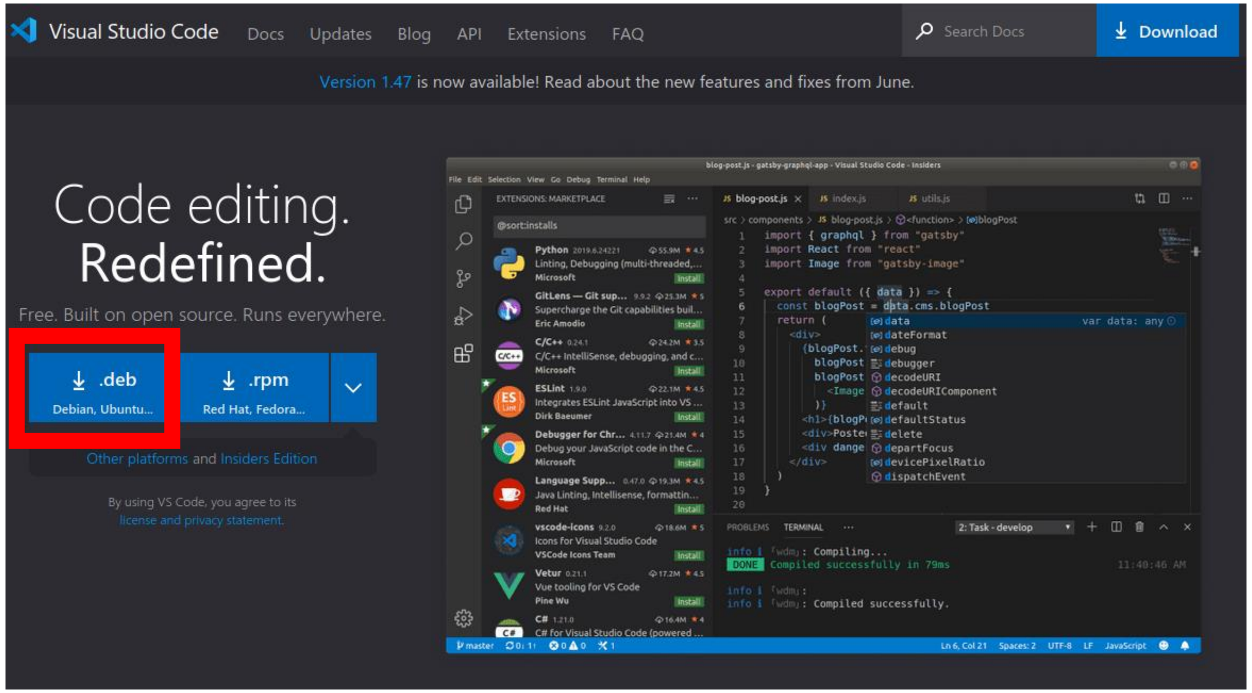


~~~
omgubuntu@ubuntu-16-10-yakkety-yak: ~/Downloads
Processing triggers for man-db (2.7.5-1) ...
omgubuntu@ubuntu-16-10-yakkety-yak:~/Downloads$ neofetch
                          ./+o+-       omgubuntu@ubuntu-16-10-yakkety-yak
                  yyyyy- -yyyyyy+      ---------------------------------
               ://+//////-yyyyyyo      Model: VMware Virtual Platform None
           .++ .:/++++++/-.+sss/`      OS: Ubuntu Yakkety Yak (development branch
         .:++o:  /++++++++/:--:/-      Kernel: 4.4.0-34-generic
        o:+o+:++.`..```.-/oo+++++/     Uptime: 2 hours, 9 minutes
       .:+o:+o/.          `+sssoo+/    Packages: 2097
  .++/+:+oo+o:`             /sssooo.   Shell: bash 4.3.46
 /+++//+:`oo+o               /::--:.   Resolution: 1440x900
 \+/+o+++`o++o               ++////.   DE: Unity
  .++.o+++oo+:`             /dddhhh.   WM: Compiz
       .+.o+oo:.          `oddhhhh+    WM Theme: Ambiance
        \+.++o+o``-````.:ohdhhhhh+     Theme: Ambiance [GTK2/3]
         `:o+++ `ohhhhhhhhyo++os:      Icons: Ubuntu-mono-dark [GTK2/3]
           .o:`.syhhhhhhh/.oo++o`      Terminal: gnome-terminal
               /osyyyyyyo++ooo+++/     CPU: Intel Core i5-2400S (1) @ 2.4GHz
                   ````` +oo+++o\:     GPU: VMware SVGA II Adapter
                          `oo++.       Memory: 561MB / 983MB

omgubuntu@ubuntu-16-10-yakkety-yak:~/Downloads$
~~~

IDE

- https://code.visualstudio.com/
- Download the *.deb* file
- Install either by:
    - Finding the file in Downloads folder and double clicking on it!
    - Or opening Terminal and typing in the following:

```
$> sudo apt install ~/Downloads/code + [TAB]
```



Version 1.47 is now available! Read about the new features and fixes from June.

Code editing. Redefined.

Free. Built on open source. Runs everywhere.

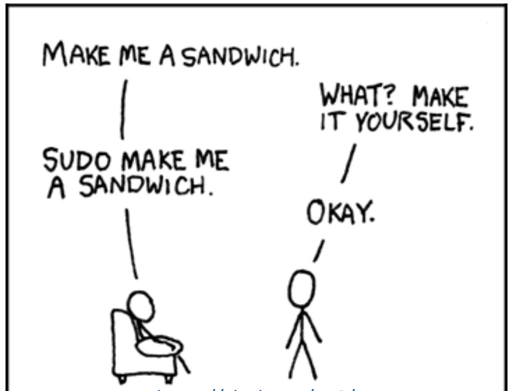
Other platforms and Insiders Edition

By using VS Code, you agree to its license and privacy statement.

```
import { graphql } from "gatsby"
import React from "react"
import Image from "gatsby-image"

export default ({ data }) => {
  const blogPost = data.cms.blogPost
  return (
```

```
info i ｢wdm｣: Compiling...
DONE Compiled successfully in 79ms
info i ｢wdm｣:
info i ｢wdm｣: Compiled successfully.
```

- **sudo** (SuperUser Do) is the command used to elevate privilege to Super User (administrator).
- **apt** (Advanced Package Tool) is a CLI program for installing and removing software in Ubuntu. **apt** takes options (optional), commands (mandatory), and arguments (optional, depending on the command) as input i.e. `apt [options] command (arguments)`. In the above example, there is no option provided, and **install** is the command telling **apt** to install the package. **install** requires an argument which is the path to the package we want to install. Running **apt** requires super user privilege.
- ~ (tilde) is the path to the user's home directory.
- Press **(TAB)** key for auto compilation.



https://xkcd.com/149/

Double Check!

- Let’s make sure we have the essential packages installed first!
  - Open a terminal and type-in the below command:

```
$> sudo apt install build-essential gdb
```

- In the above command the argument provided to the **install** command is not a path to a local file. Thus, **apt** will look in the registered repositories for a package with the same name to fetch and install them.
- **build-essential** is a package containing GNU’s C/C++ compilers and libraries, and more development tools and libraries. For more details, please visit https://packages.ubuntu.com/xenial/build-essential.
- **gdb** is the GNU C/C++ debugger.

Workspace

- Usually, the folder where all the project artifacts reside. Projects are most often synced with a revision control tools.

- IDEs create files in workspace folders to save project related settings (more to come on this, in future lectures!)

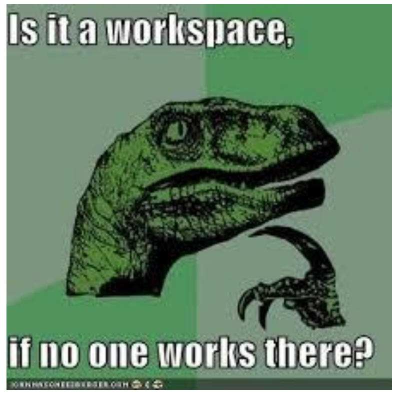
- Create a folder in you home directory and call it `projects`
  - You can create a folder either through file explorer or terminal:

```
$> mkdir ~/projects
```

- Create a folder for our coming project `HelloWorld` and add that folder to your VSCode workspace:
  - You can do this either through VSCode menus or through terminal:

```
$> mkdir ~/projects/HelloWorld
$> cd ~/projects
$> code .
```

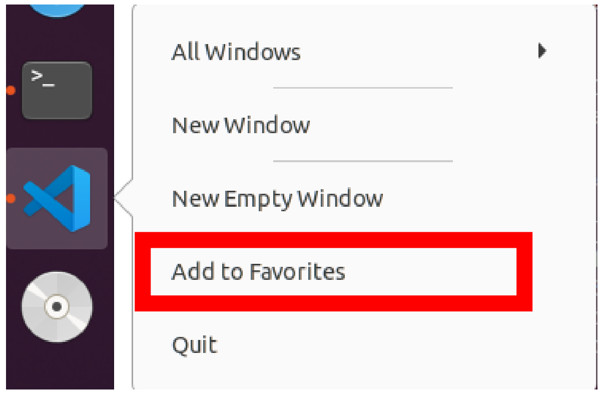
  - This will open an instance of VSCode for our currently empty workspace.

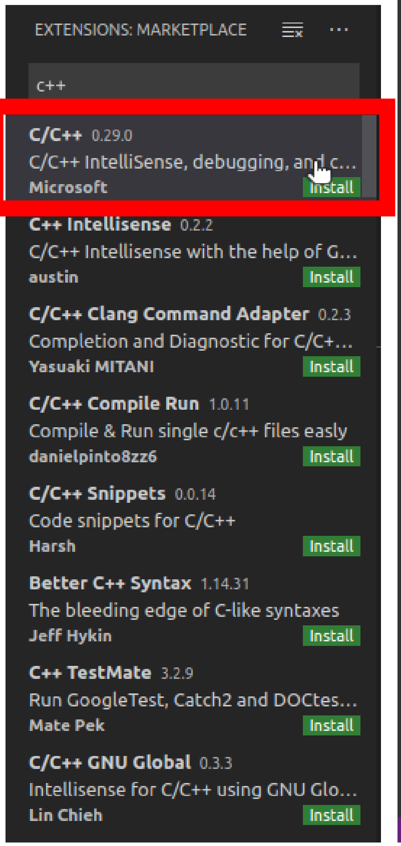


- **mkdir** (MaKe DIRectory) creates the folder provided as its argument if the user running the command has privileges to create the folder on the specified path, such as user's home directory.
- **code** is the Visual Studio Code and opens an instance on the provided path and makes it a workspace, if it is not already.
- At any stage while typing in commands in the terminal, one could use (TAB) key for suggestions on compilation of the command.

Visual Studio Code

- Extension based IDE with a marketplace for add-ons
- Click on "*Tools and Languages*" or `[Ctrl+Shft+x]`
- Install C/C++ extension by Microsoft




EXTENSIONS: MARKETPLACE

c++

C/C++ 0.29.0
C/C++ IntelliSense, debugging, and c…
Microsoft Install

C++ Intellisense 0.2.2
C/C++ Intellisense with the help of G…
austin Install

C/C++ Clang Command Adapter 0.2.3
Completion and Diagnostic for C/C+…
Yasuaki MITANI Install

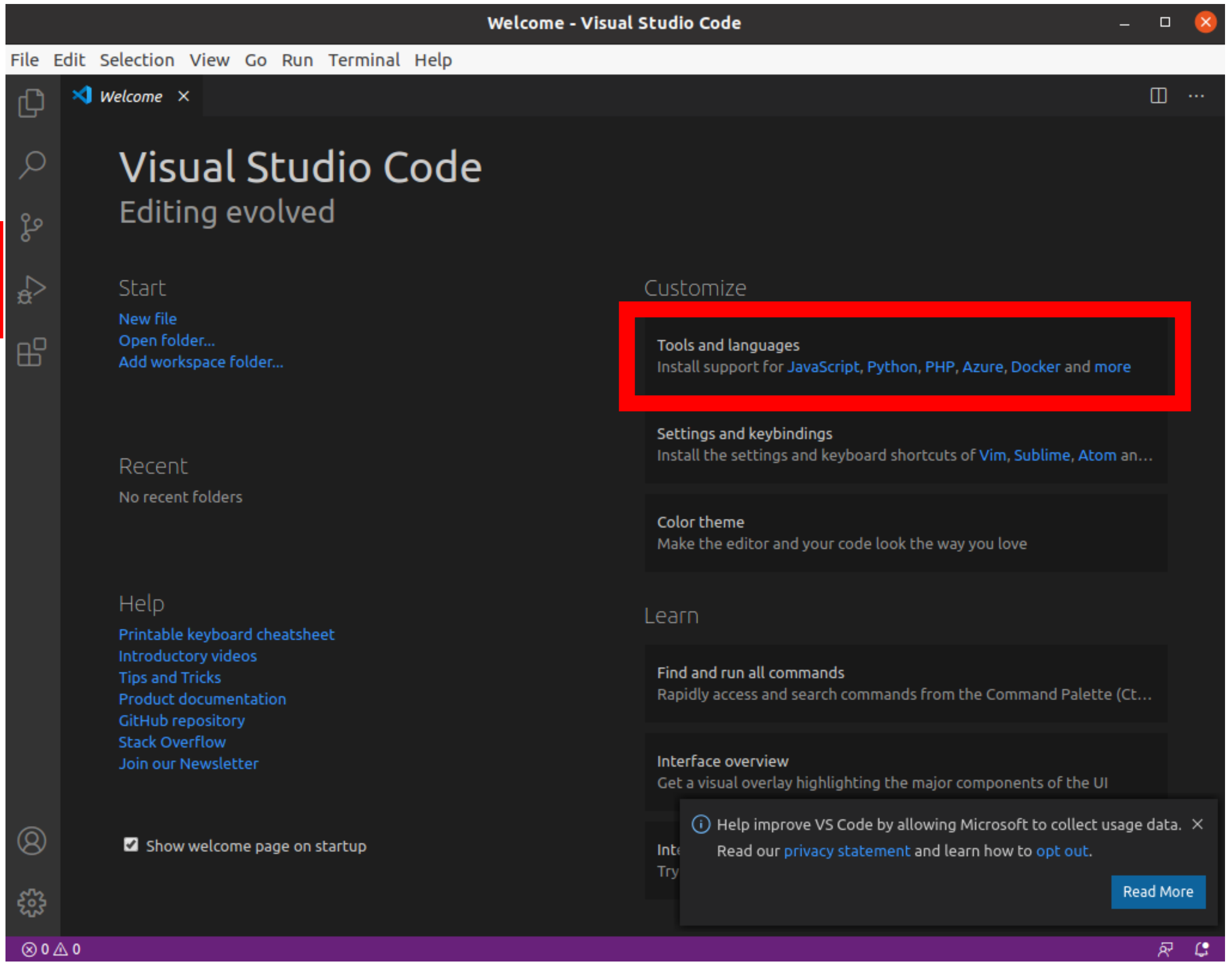
C/C++ Compile Run 1.0.11
Compile & Run single c/c++ files easly
danielpinto8zz6 Install

C/C++ Snippets 0.0.14
Code snippets for C/C++
Harsh Install

Better C++ Syntax 1.14.31
The bleeding edge of C-like syntaxes
Jeff Hykin Install

C++ TestMate 3.2.9
Run GoogleTest, Catch2 and DOCtes…
Mate Pek Install

C/C++ GNU Global 0.3.3
Intellisense for C/C++ using GNU Glo…
Lin Chieh Install


Welcome - Visual Studio Code

File Edit Selection View Go Run Terminal Help

Welcome ×

Visual Studio Code
Editing evolved

Start
New file
Open folder...
Add workspace folder...

Recent
No recent folders

Help
Printable keyboard cheatsheet
Introductory videos
Tips and Tricks
Product documentation
GitHub repository
Stack Overflow
Join our Newsletter

☑ Show welcome page on startup

Customize

Tools and languages
Install support for JavaScript, Python, PHP, Azure, Docker and more

Settings and keybindings
Install the settings and keyboard shortcuts of Vim, Sublime, Atom an…

Color theme
Make the editor and your code look the way you love

Learn

Find and run all commands
Rapidly access and search commands from the Command Palette (Ct…

Interface overview
Get a visual overlay highlighting the major components of the UI

Help improve VS Code by allowing Microsoft to collect usage data. ×
Read our privacy statement and learn how to opt out.
Read More

⊗ 0 ⚠ 0

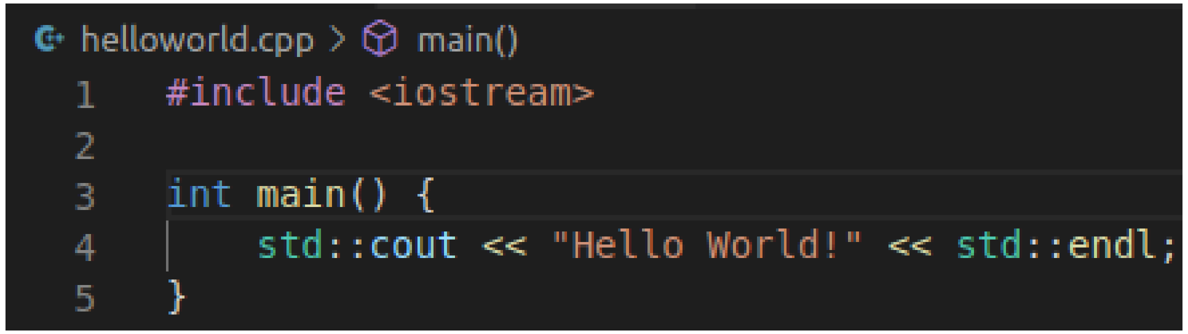
- All the keyboard shortcuts could be found here: https://code.visualstudio.com/shortcuts/keyboard-shortcuts-linux.pdf
- Once you opened VSCode, right click on the logo and add to favorite for easier access in future.

Hello World!

- Create a new file `[Ctrl+N]` and paste the following:

```
#include <iostream>

int main() {
    std::cout << "Hello World!" << std::endl;
}
```

- Save the file `[Ctrl+S]` and name the file `helloworld.cpp` – make sure the file is in `HelloWorld` folder. Once you save the file, *syntax highlighting* should be enabled for you, as depicted below.

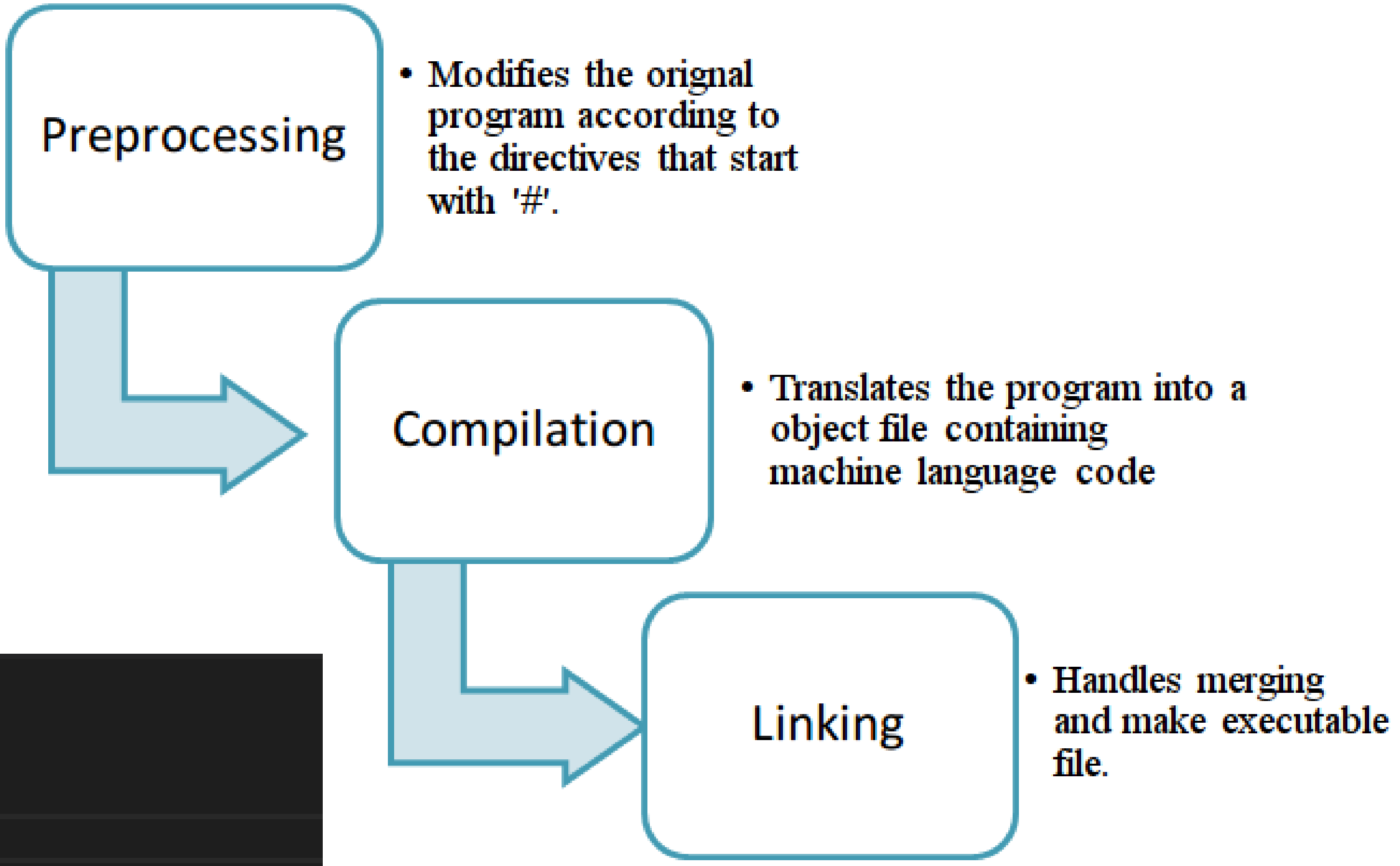

helloworld.cpp > main()

```
#include <iostream>

int main() {
    std::cout << "Hello World!" << std::endl;
}
```

- cpp is a file extension used for C++ source code.

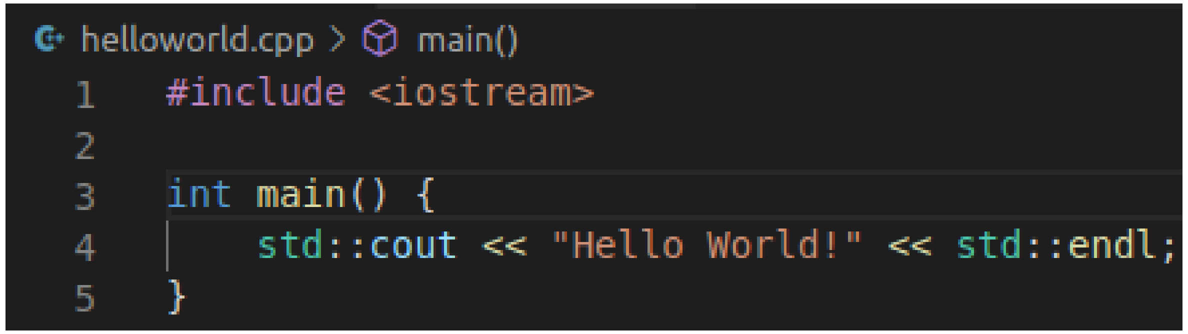
Turning Text To Binary – In Theory

- Software Building Procedure



- Modifies the orignal program according to the directives that start with '#'.
- Translates the program into a object file containing machine language code
- Handles merging and make executable file.

http://www.cplusplus.com/articles/2v07M4Gy/Selection_101.png



```cpp
#include <iostream>

int main() {
    std::cout << "Hello World!" << std::endl;
}
```

Turning Text To Binary – In Action

- Open a terminal and navigate to the project directory:

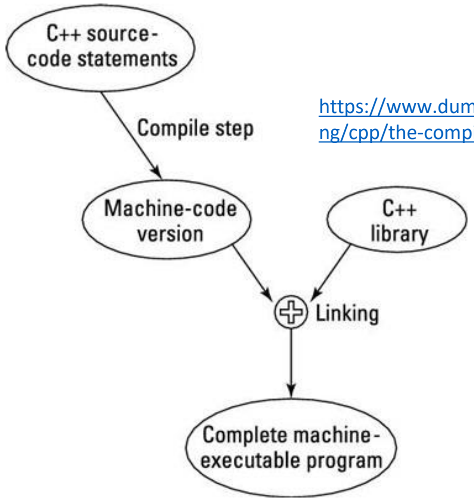
```
$> cd ~/projects/HelloWorld
```

- Compile the cpp file and name the output binary `hw`:

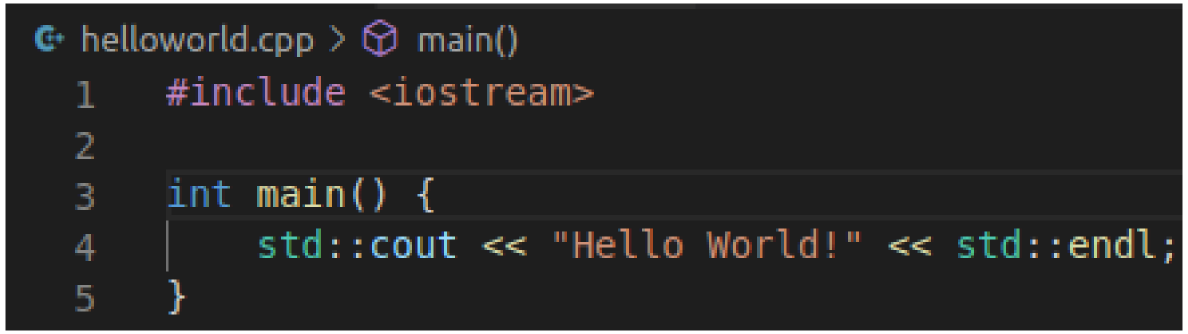
```
$> g++ helloworld.cpp -o hw
```

- Run the executable:

```
$> ./hw
```


C++ source-code statements
Compile step
Machine-code version
C++ library
Linking
Complete machine-executable program

https://www.dummies.com/programming/cpp/the-compilation-process-in-c/



```cpp
#include <iostream>

int main() {
    std::cout << "Hello World!" << std::endl;
}
```

- **cd** (Change Directory) is the command for navigating to a different path. The path one wants to navigate to is provided as the argument to **cd**.
- **g++** is the GNU C++ compiler. It takes as input the source codes (.cpp files). It also receives *flags*; these are the options which has **– (hyphen/dash)** as prefix. Here **–o** *flag* indicates the name of the output binary and it requires an argument which is the name of the binary.
- Executables could be executed by invoking the path to them. In the above example, since we are in the folder where the binary exists, we use **. (dot)** which points to the current directory we are at.

C++ Source Code Structure

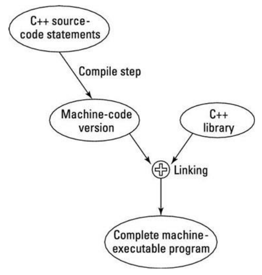
- g++ knows stuff!

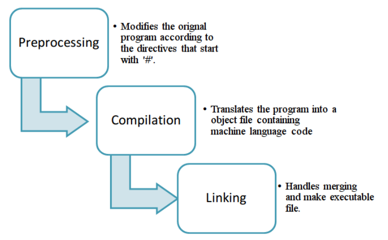
```
$> g++ helloworld.cpp -o hw
```

  - Keywords and symbols
  - Routine
  - Machine language (binary)

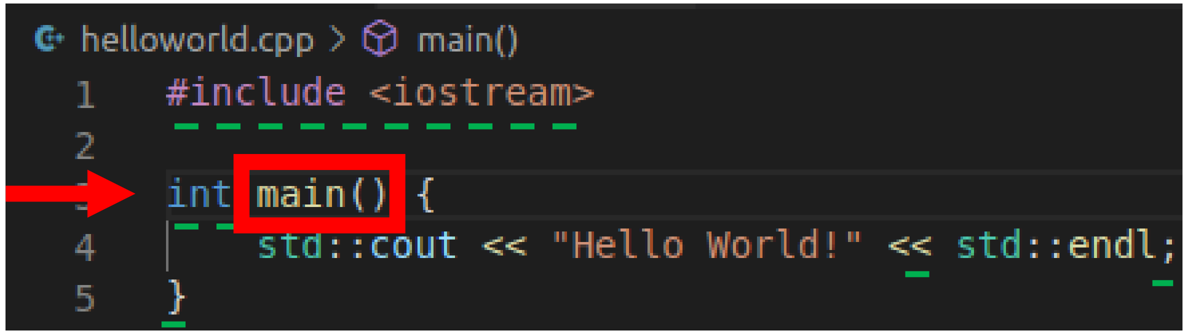
- g++ generates error if things are not as it expects them to be!

- g++ reads source code line by line from the beginning of the file


C++ source-code statements
Compile step
Machine-code version
C++ library
Linking
Complete machine-executable program


Preprocessing
• Modifies the orignal program according to the directives that start with '#'.
Compilation
• Translates the program into a object file containing machine language code
Linking
• Handles merging and make executable file.

- Each line in source code starts with an identifiers. Identifiers either start with **#** (number sign) for preprocessing, or by a letter for compilation. Identifiers cannot start with digits nor characters, except for **_** (underscore / underline).
- Lines starting with **//** double slashes are comments and not considered by the compiler; same as the section of the code between **/*** and ***/**



helloworld.cpp > main()

```cpp
#include <iostream>

int main() {
    std::cout << "Hello World!" << std::endl;
}
```

```
#Some_PreProcessing_Stuff


function1_signature() {
// FUNCTION BODY
}
```

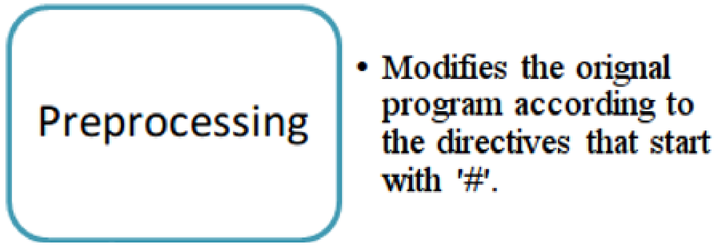
C++ Preprocessing

- Procedure prior to compilation performed by preprocessor
  - Gives the opportunity to impact compilation
  - Preprocessor prepares source code for compiler

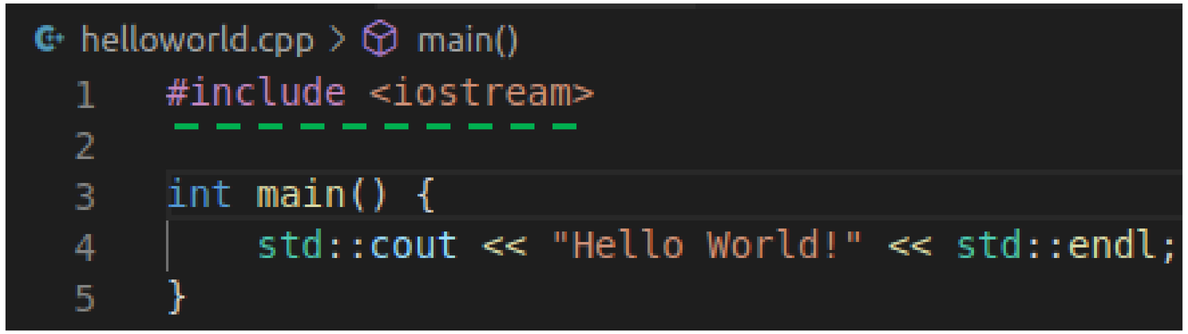
- Preprocessor
  - Operators: **#**, #@, ##, //, /**/

  - Directives
    - They start with **#**: #define, #error, #import, #undef, #elif, #if, **#include**, #using, #else, #ifdef, #line, #endif, #ifndef, #pragma

"The preprocessor is a text processor that manipulates the text of a source file [...] the compiler ordinarily invokes the preprocessor in its first pass, the preprocessor can also be invoked separately to process text without compiling". https://docs.microsoft.com/en-us/cpp/preprocessor/preprocessor?view=vs-2019



- Modifies the orignal program according to the directives that start with '#'.



```cpp
#include <iostream>

int main() {
    std::cout << "Hello World!" << std::endl;
}
```

- **#include** directives copies the whole contents of the file it is provided at the exact point the directive is used. The operation could be recursive for nested includes i.e. when a file which is included has its own **#include** directive. It is common practice for better readability and maintainability to always used this directive at the very beginning of the files.

Exercises !

- Build and Execute Hello World Program.
- Test Compilation Error Scenario.

C++ Operators

- Almost all the special characters are used in C++
- Same characters might operate differently in different contexts
- Assume X=30 and Y=10.

- **=** is the assignment operator in C++.

Arithmetic operator	Description	Example
+	addition	X+Y=40
−	subtraction	X-Y=20
*	multiplication	X*Y=300
/	division	X/Y=3
%	Modulo(Reminder after an integer division)	X%Y=0

Compound assignment	Description	Example
+=	addition and assignment	y += x; so y = y + x;
-=	subtraction and assignment	x -= 3; so x=x-3;
/=	multiplication and assignment	y /= x; so y=y/x;
*=	division and assignment	y*=x; so y=y*x;
%=	modulo and assignment	y%=x; so y=y%x;
++	increasing by one unit	y++ =11
--	Decreasing by one unit	y--=9

Comparison operator	Description
==	equal to
!=	not equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

Logical operator	Description
!	Logical Not
&&	Logical And
||	Logical OR

Check BitWise Operators

Precedence of Operators

Level	Precedence group	Operator	Description	Grouping
1	Scope	`::`	scope qualifier	Left-to-right
2	Postfix (unary)	`++ --`	postfix increment / decrement	Left-to-right
		`()`	functional forms	
		`[]`	subscript	
		`. ->`	member access	
3	Prefix (unary)	`++ --`	prefix increment / decrement	Right-to-left
		`~ !`	bitwise NOT / logical NOT	
		`+ -`	unary prefix	
		`& *`	reference / dereference	
		`new delete`	allocation / deallocation	
		`sizeof`	parameter pack	
		`(type)`	C-style type-casting	
4	Pointer-to-member	`.* ->*`	access pointer	Left-to-right
5	Arithmetic: scaling	`* / %`	multiply, divide, modulo	Left-to-right
6	Arithmetic: addition	`+ -`	addition, subtraction	Left-to-right
7	Bitwise shift	`<< >>`	shift left, shift right	Left-to-right
8	Relational	`< > <= >=`	comparison operators	Left-to-right
9	Equality	`== !=`	equality / inequality	Left-to-right
10	And	`&`	bitwise AND	Left-to-right
11	Exclusive or	`^`	bitwise XOR	Left-to-right
12	Inclusive or	`|`	bitwise OR	Left-to-right
13	Conjunction	`&&`	logical AND	Left-to-right
14	Disjunction	`||`	logical OR	Left-to-right
15	Assignment-level expressions	`= *= /= %= += -= >>= <<= &= ^= |=`	assignment / compound assignment	Right-to-left
		`?:`	conditional operator	
16	Sequencing	`,`	comma separator	Left-to-right

C++ Keywords

- Fundamental Datatypes
- Control Flows: Selections, Iteration Statements & Jump Statements

alignas, alignof, and, and_eq, asm, auto, bitand, bitor, **bool**, **break**, **case**, catch, **char**, **char16_t**, **char32_t**, class, compl, **const**, constexpr, const_cast, **continue**, **decltype**, **default**, delete, **do**, **double**, dynamic_cast, **else**, enum, explicit, export, extern, **false**, **float**, **for**, friend, **goto**, **if**, inline, **int**, **long**, mutable, namespace, new, noexcept, not, not_eq, **nullptr**, operator, or, or_eq, private, protected, public, register, reinterpret_cast, **return**, **short**, **signed**, **sizeof**, static, static_assert, static_cast, struct, **switch**, template, this, thread_local, throw, **true**, try, typedef, typeid, typename, union, **unsigned**, **using**, virtual, **void**, volatile, **wchar_t**, **while**, xor, xor_eq

- Identifiers created by developers shall not match these keywords.
- Different compilers might add specific keywords.
- C++ is **case-sensitive**.

Fundamental Datatypes

- Datatypes define the type of data.
- Could be used to define variables
  - Variables are declared as below:

```
SomeDatatype VariableName;
```

- Apart from the fundamental datatypes, there are other datatypes which we will discuss later. Developers can also create their custom datatypes, as well.
- Variable names are identifiers and the same restrictions for the names apply.
- **signed** keyword is not necessary, an integer is considered signed unless it is declared otherwise using **unsigned** keyword.

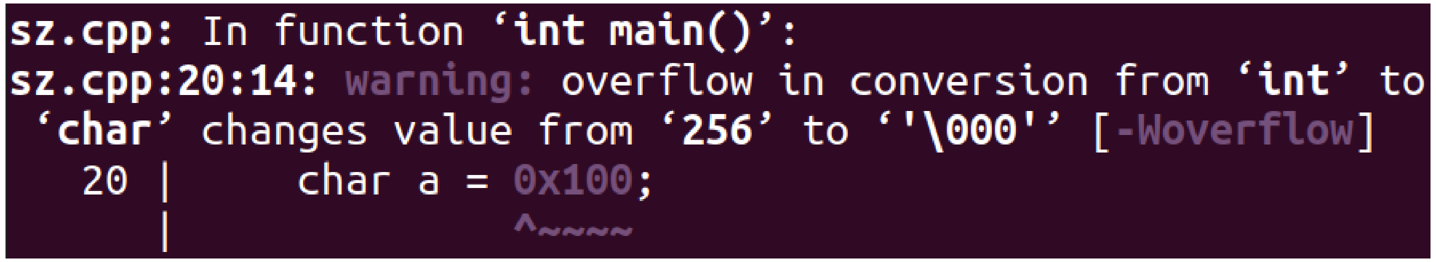
Character types	**char**
	char16_t
	char32_t
	wchar_t
Floating-point types	**float**
	double
	long double
Boolean type	**bool**
Void type	void
Null pointer	decltype(nullptr)

Integer types (signed)	signed char
	signed short int
	signed **int**
	signed long int
	signed long long int
Integer types (unsigned)	unsigned char
	unsigned short int
	unsigned int
	unsigned long int
	unsigned long long int

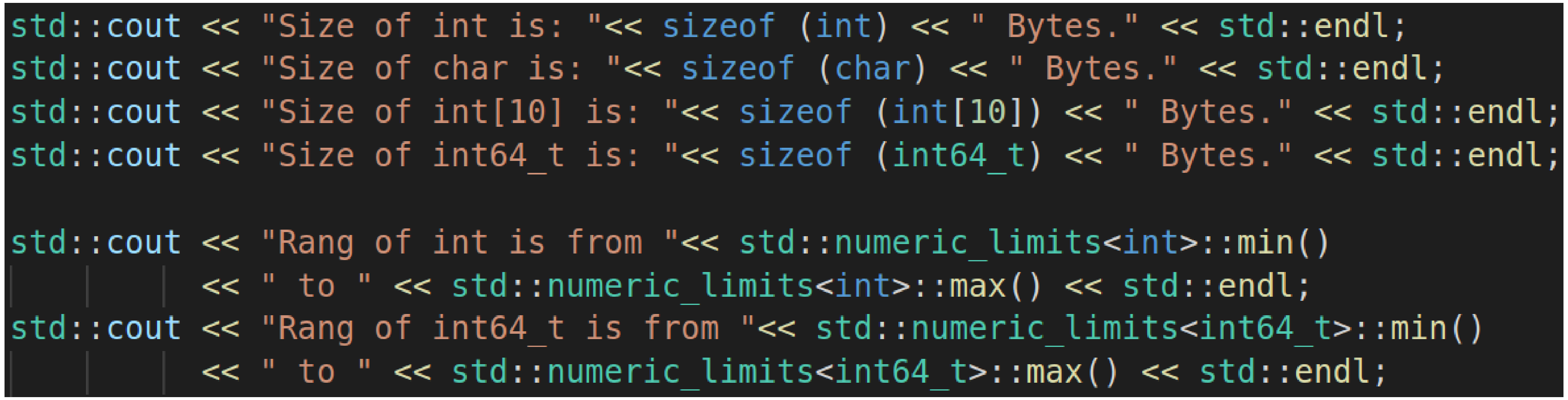
Size, Range & Overflow

- Fundamental datatypes have a range.

- Value higher than range is *overflow*.
- Why integer size varies from computer to computer?



```
sz.cpp: In function ‘int main()’:
sz.cpp:20:14: warning: overflow in conversion from ‘int’ to
 ‘char’ changes value from ‘256’ to ‘'\000'’ [-Woverflow]
   20 |     char a = 0x100;
      |              ^~~~~
```

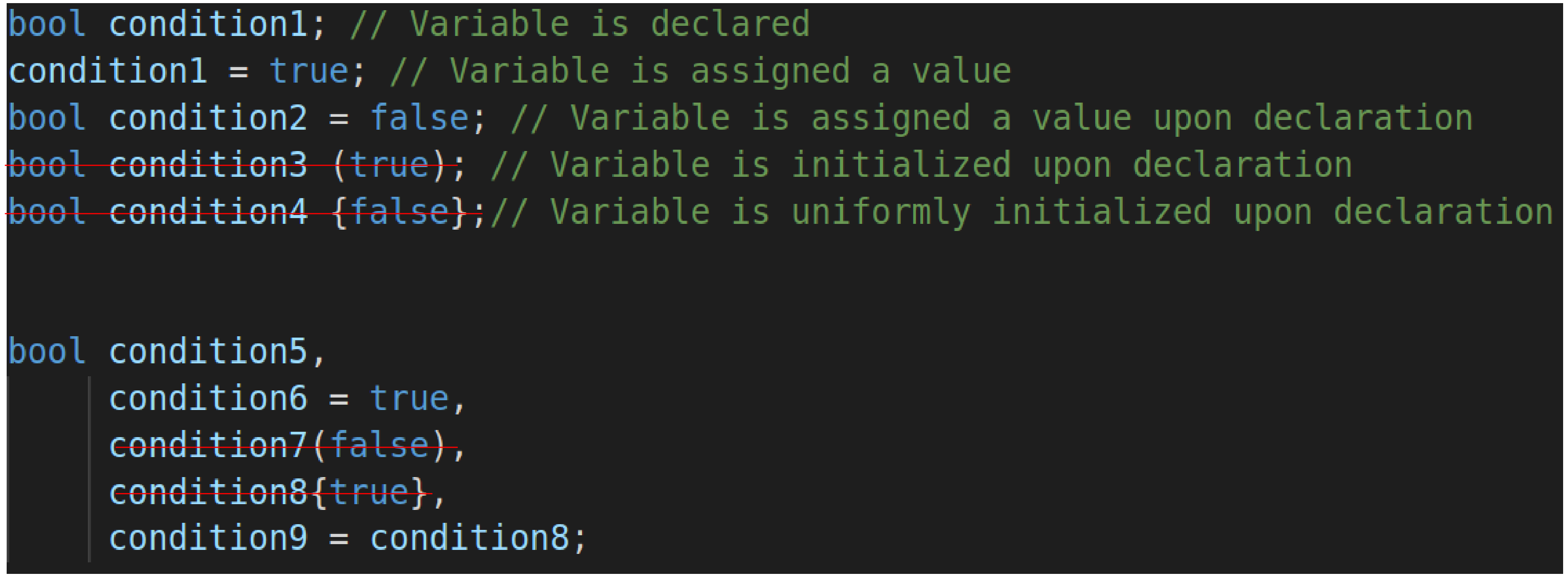


```
std::cout << "Size of int is: "<< sizeof (int) << " Bytes." << std::endl;
std::cout << "Size of char is: "<< sizeof (char) << " Bytes." << std::endl;
std::cout << "Size of int[10] is: "<< sizeof (int[10]) << " Bytes." << std::endl;
std::cout << "Size of int64_t is: "<< sizeof (int64_t) << " Bytes." << std::endl;

std::cout << "Rang of int is from "<< std::numeric_limits<int>::min()
          << " to " << std::numeric_limits<int>::max() << std::endl;
std::cout << "Rang of int64_t is from "<< std::numeric_limits<int64_t>::min()
          << " to " << std::numeric_limits<int64_t>::max() << std::endl;
```

Variables

- Variables could be assigned a value of their type.

- Be careful, C++ is partially type-safe!
  - If a variable is assigned a value of different type, the compiler might not generate an error!
  - There are safe mechanism for type conversions -- more on this in future lectures.



```
bool condition1; // Variable is declared
condition1 = true; // Variable is assigned a value
bool condition2 = false; // Variable is assigned a value upon declaration
~~bool condition3 (true);~~ // Variable is initialized upon declaration
~~bool condition4 {false};~~// Variable is uniformly initialized upon declaration


bool condition5,
     condition6 = true,
     ~~condition7(false)~~,
     ~~condition8{true}~~,
     condition9 = condition8;
```

- **true** and **false** are the two keywords defined in C++ for Boolean values. **true** is equal to 1 and **false** is 0: https://stackoverflow.com/questions/2725044/can-i-assume-booltrue-int1-for-any-c-compiler
- Initializing a variable and assigning a value upon declaration are VERY similar but they are **not** the exact same thing! We discuss this in detail in the coming lectures.

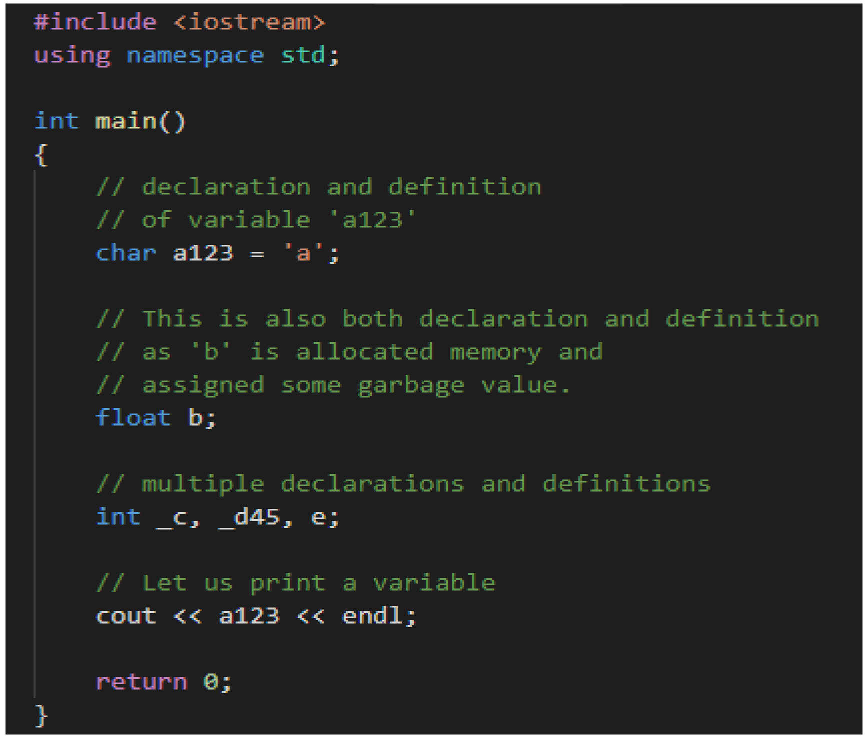
Variables

- The variable declaration refers to the part where a variable is first declared or introduced before its first use. A variable definition is a part where the variable is assigned a memory location and a value. Most of the times, variable declaration and definition are done together.

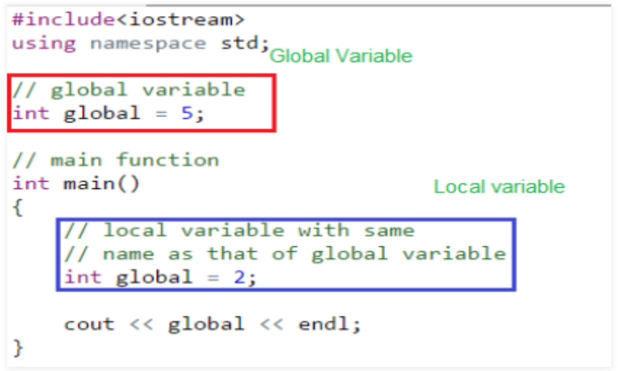
- **Variable Scope:**

1.Block or Function Scope(Local variables)
2.File Scope (Global Variables)
3.Project/Program Scope (Externed Global Variable using extern keyword and it used by different files).

- **Types of variables**



```
#include <iostream>
using namespace std;

int main()
{
    // declaration and definition
    // of variable 'a123'
    char a123 = 'a';

    // This is also both declaration and definition
    // as 'b' is allocated memory and
    // assigned some garbage value.
    float b;

    // multiple declarations and definitions
    int _c, _d45, e;

    // Let us print a variable
    cout << a123 << endl;

    return 0;
}
```



```
#include<iostream>
using namespace std;

// global variable
int global = 5;

// main function
int main()
{
    // local variable with same
    // name as that of global variable
    int global = 2;

    cout << global << endl;
}
```

Variables in C++ - GeeksforGeeks

Variables

- Variable Storage Duration/Life Time:
  - Local Allocation:

Variables are created and the memory is allocated in the stack upon entering the function or the block and they are destroyed upon exiting the function or the block.

  - Global and Static Allocation:

Variables exist in the memory during the entire execution of the program. The memory is allocated before the run time.

  - Dynamic Allocation:

Memory allocation during the run time in the heap.The programmer controls the life time of the variable.

- Variables Storage Classes:
  - C++ uses 5 storage classes, namely:

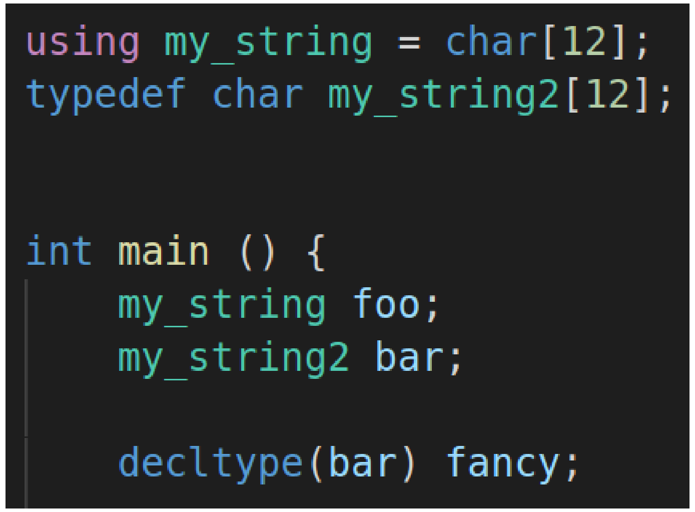
1. auto
2. register
3. extern
4. static
5. mutable

Exercises !

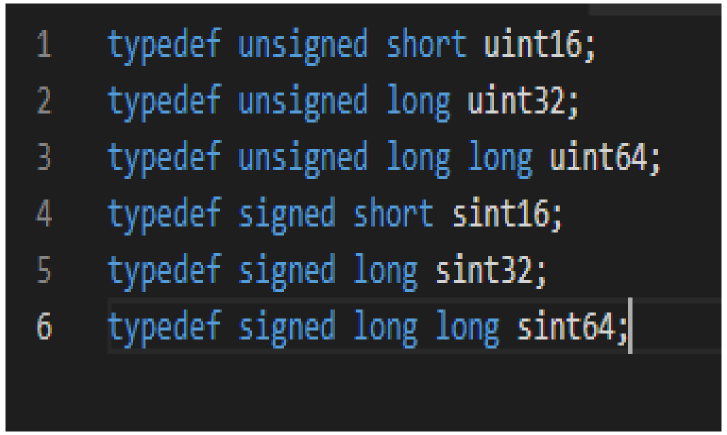
- Write a program for converting temperature from degrees Celsius to degrees Fahrenheit.(EX1.3)
- Write a program to print the ASCII value of a character input by the user.(EX1.5)

Types

- Type alias is with `using` keyword.
- With `typedef` you can create synonym for a type.
- It is also possible to detect type of a variable at run time using `decltype`.



```
using my_string = char[12];
typedef char my_string2[12];


int main () {
    my_string foo;
    my_string2 bar;

    decltype(bar) fancy;
```



```
typedef unsigned short uint16;
typedef unsigned long uint32;
typedef unsigned long long uint64;
typedef signed short sint16;
typedef signed long sint32;
typedef signed long long sint64;
```

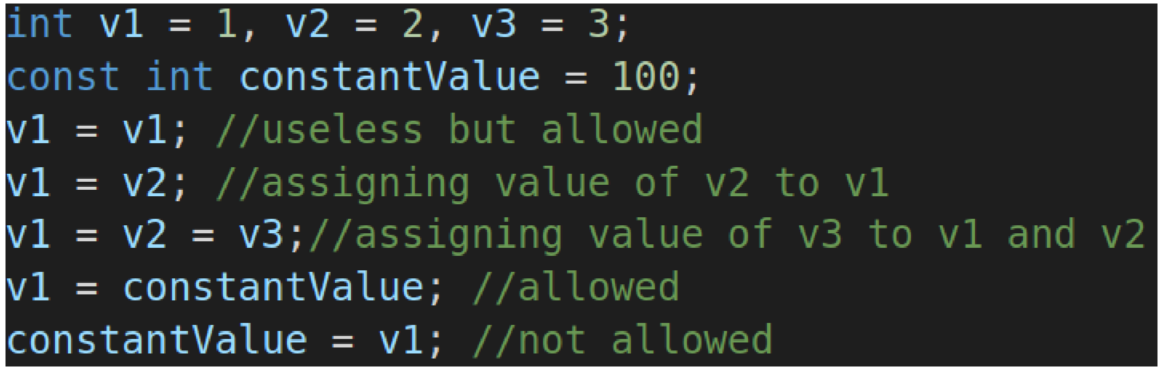
Constants

- Constant Variables are declared as below:

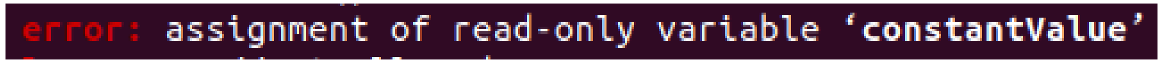
```
const SomeDatatype VariableName
```

- Constant variables are read-only

- Constant variables shall be assigned a value/initialized upon declaration



```
int v1 = 1, v2 = 2, v3 = 3;
const int constantValue = 100;
v1 = v1; //useless but allowed
v1 = v2; //assigning value of v2 to v1
v1 = v2 = v3;//assigning value of v3 to v1 and v2
v1 = constantValue; //allowed
constantValue = v1; //not allowed
```



```
error: assignment of read-only variable ‘constantValue’
```

- Although there are myths that using const might improve performance, since the compiler could optimize the code better (which in some rare cases is true), the main reason for declaring const variables are maintainability and enforcing correctness: https://stackoverflow.com/questions/3435026/can-const-correctness-improve-performance
- Compiler generates an error and terminates compilation if an attempt is made to modify a **const** value after declaration.

Arrays

- Arrays are declared using `[]`
  `SomeDatatype VariableName[]`
- Arrays are NOT dynamic in size.
- Size of the array should be known to the compiler upon declaration.
- Array name is a constant pointer holds the first element address.

```
int twoDimensionArray[2][2];
twoDimensionArray[0][0] = 432;
int threeDimensionArray[2][2][2] = {
    {{1,2},{3,4}},
    {{5,6},{7,8}}
};
```

- Array's elements could be accessed via their indices – arrays are index from 0 i.e. for an array of size N, the first element is at index 0 and the last element is stored in index N-1.
- If the size of the array is not defined upon declaration, compiler generates an error and terminates compilation.

```
error: storage size of ‘arrayOfIntegers_3’ isn’t known
```

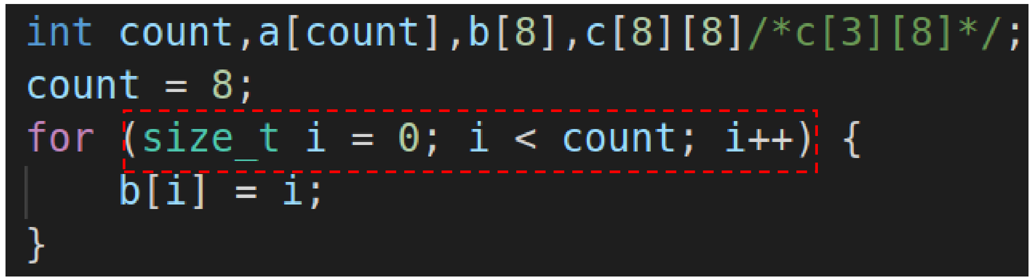
```
unsigned int arrayOfIntegers_3[]; // not allowed, the size is needed
unsigned int arrayOfIntegers_2[] = {1,2,3};// array of size 3 is declared and initialized upon declaration
unsigned int arrayOfIntegers_1[3];// array of size 3 is declared
arrayOfIntegers_1[0] = 53453; // first element of the array is assigned a value
arrayOfIntegers_1[1] = 29614; // second element of the array is assigned a value
arrayOfIntegers_1[2] = arrayOfIntegers_2[2]; // third element of the array is assigned a value
```

Arrays

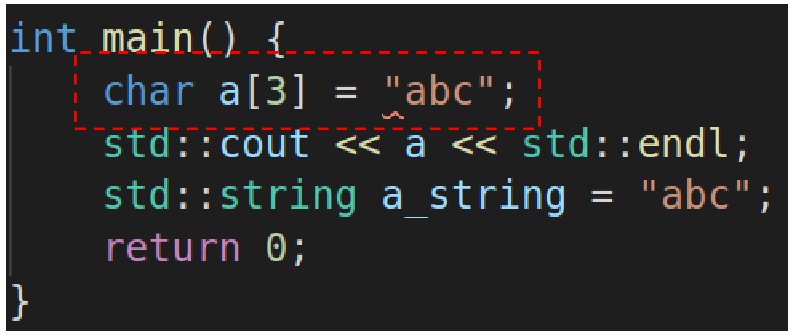
- If the number of array initializers is less than the array size, the remaining array elements are initialized to zero.
- Suppose, there is array of 12 elements(arr[12]).You can use array elements from arr[0] to arr[11].So what if you try to use arr[12] or arr[15] ?. The compiler may not show error but fatal error will happen during program execution.
- If the number of initializers are larger than the number of array elements this will cause a compiler error. For example, Int arr[2]={5,9,10} (error).
- Multidimensional array(array of array) is abstraction for the developer, as the same result can be achieved by 1D array. For example, int arr[2][3] is equivalent to int arr[6].
- Check different search algorithms here.

Arrays & Strings

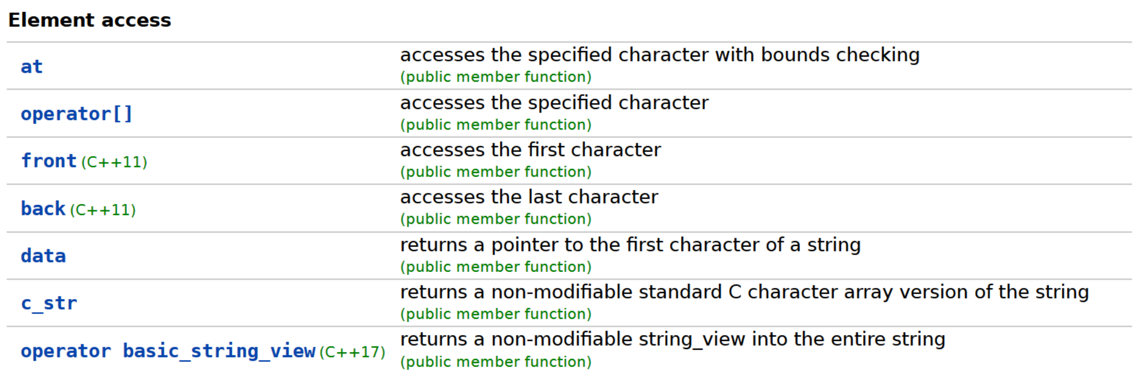
- Strings are 1D array of characters, this array is terminated by null. For strings manipulations, there are several functions(such as strlen(), strcpy(),strcat(), strcmp(),strlwr(),and strupr()) in “string.h” header file.
- There is no check on arrays’ boundaries.
- Strings are ASCI code null terminated.
- Standard library provides `string`.



```
int count,a[count],b[8],c[8][8]/*c[3][8]*/;
count = 8;
for (size_t i = 0; i < count; i++) {
    b[i] = i;
}
```



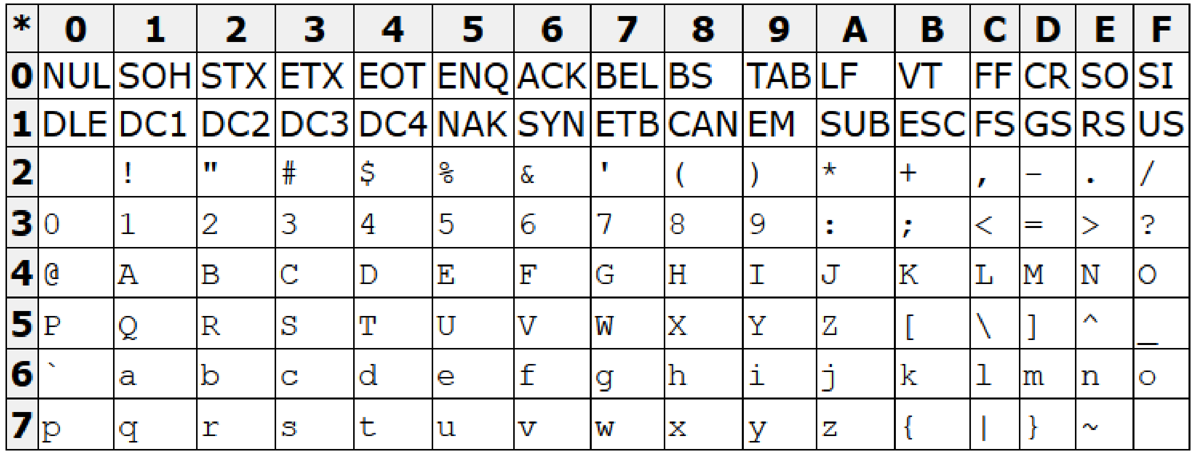
```
int main() {
    char a[3] = "abc";
    std::cout << a << std::endl;
    std::string a_string = "abc";
    return 0;
}
```



Element access

`at`	accesses the specified character with bounds checking (public member function)
`operator[]`	accesses the specified character (public member function)
`front` (C++11)	accesses the first character (public member function)
`back` (C++11)	accesses the last character (public member function)
`data`	returns a pointer to the first character of a string (public member function)
`c_str`	returns a non-modifiable standard C character array version of the string (public member function)
`operator basic_string_view` (C++17)	returns a non-modifiable string_view into the entire string (public member function)

https://en.cppreference.com/w/cpp/string/basic_string



*	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

http://www.cplusplus.com/doc/ascii/

Control Flows: Selection

- They choose (executed) different paths (branches) depending on whether some conditions holds:
  - **`switch`** checks whether the expression matches any of the cases
  - **`if/else`** checks whether the condition holds
  - Conditional Operator is a shorthand notation for if/else statement.

> - It is considered a very good practice to always have a default case in our switch statements, although the need might not seem reasonable in some scenarios: https://stackoverflow.com/questions/4649423/should-switch-statements-always-contain-a-default-clause

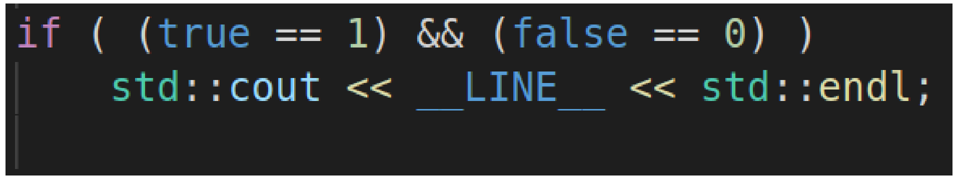
```
switch (expression)
{
case constantExpression1:
    /* code */
    break;
case constantExpression2:
    /* code */
    break;
case constantExpression3:
    /* code */
    break;
default:
    break;
}
```

```
if (someCondition) {
    // DO SOMETHING
}

if (someCondition) {
    // DO SOMETHING
} else {
    // DO SOMETHING ELSE
}

if (someCondition) {
    // DO SOMETHING
} else if (anotherCondition) {
    // DO SOMETHING ELSE
} else {
    // DO NONE OF THE ABOVE
}
```

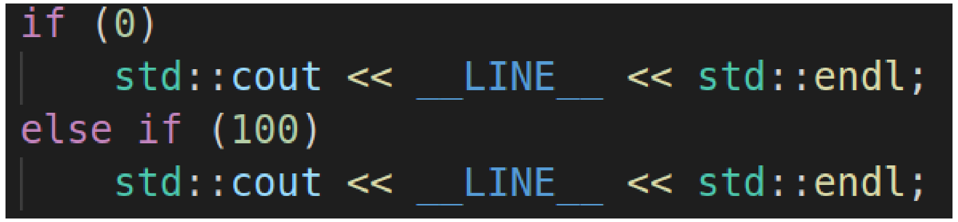
if

Boolean Value	Operand	Boolean Value	Result
true	&&	true	true
true	&&	false	false
false	&&	false	false
false	&&	true	false
true	||	true	true
true	||	false	true
false	||	false	false
false	||	true	true

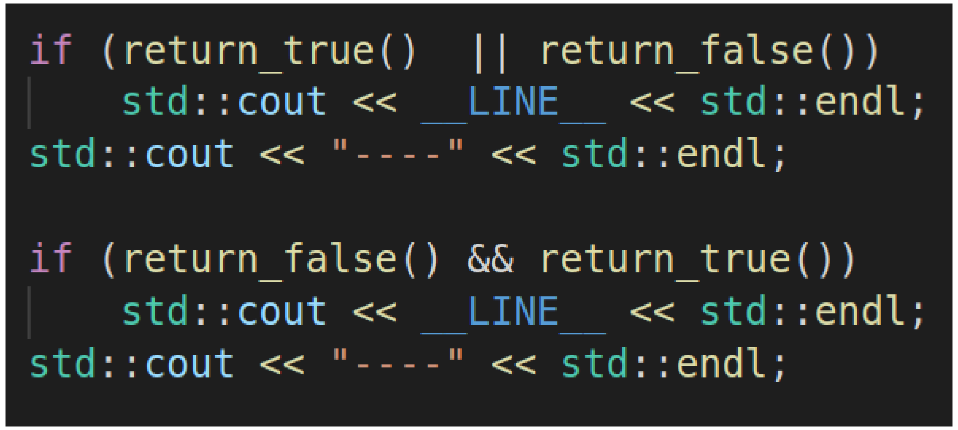
- **__LINE__** is a preprocessor Macro which *expands* to the line number. There are other useful Macros as well: https://stackoverflow.com/a/2849850



```
if ( (true == 1) && (false == 0) )
    std::cout << __LINE__ << std::endl;
```

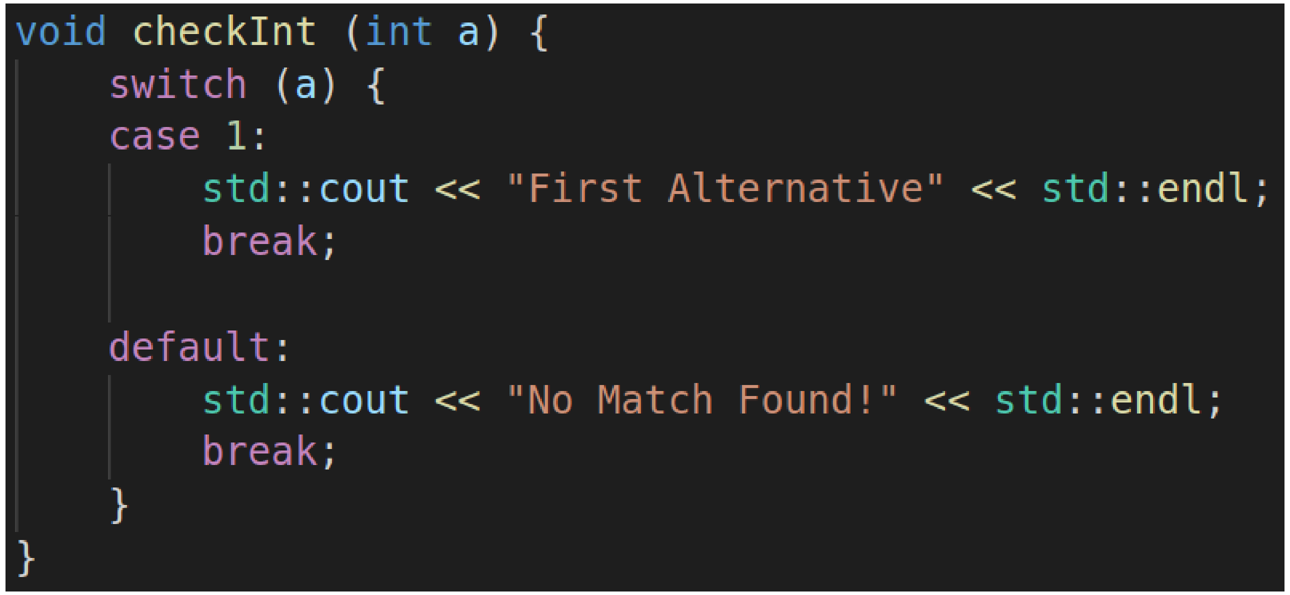


```
if (0)
    std::cout << __LINE__ << std::endl;
else if (100)
    std::cout << __LINE__ << std::endl;
```

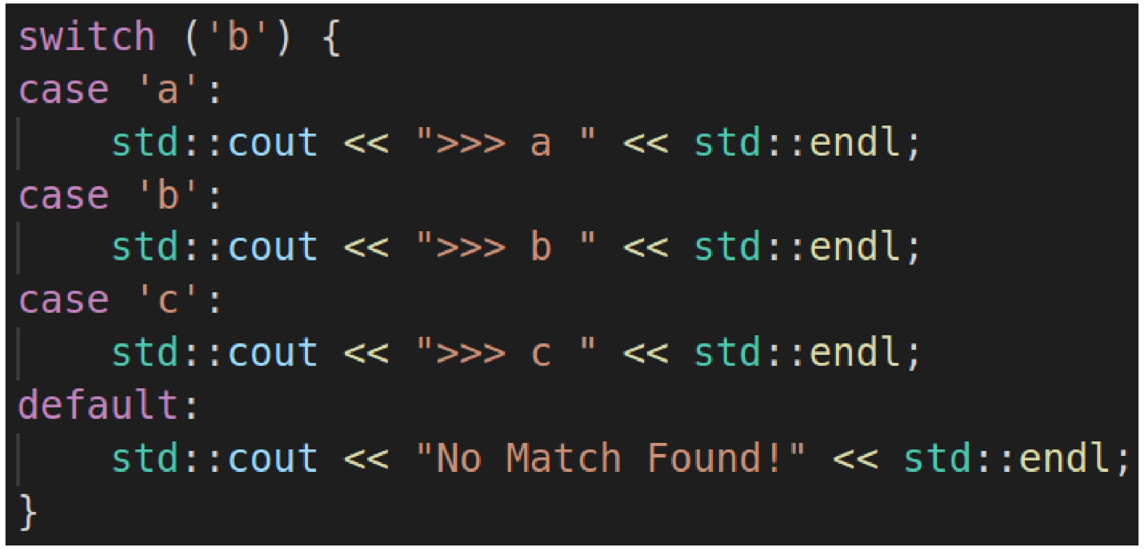


```
if (return_true()  || return_false())
    std::cout << __LINE__ << std::endl;
std::cout << "----" << std::endl;

if (return_false() && return_true())
    std::cout << __LINE__ << std::endl;
std::cout << "----" << std::endl;
```

switch



```
void checkInt (int a) {
    switch (a) {
    case 1:
        std::cout << "First Alternative" << std::endl;
        break;

    default:
        std::cout << "No Match Found!" << std::endl;
        break;
    }
}
```



```
switch ('b') {
case 'a':
    std::cout << ">>> a " << std::endl;
case 'b':
    std::cout << ">>> b " << std::endl;
case 'c':
    std::cout << ">>> c " << std::endl;
default:
    std::cout << "No Match Found!" << std::endl;
}
```

Control Flows: Iteration Statements

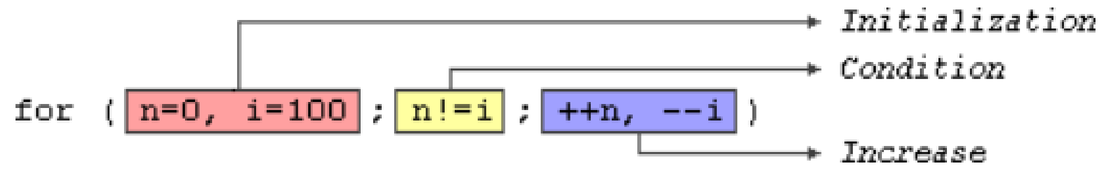
- They Iterate (loop) over some section of the code as long as a condition is **`true`**:
  - **`while`** iterates as long as provided condition holds.
  - **`do/while`** executes the segment provided to **`do`**, then checks for the condition in **`while`**, repeats as long as provided condition holds.
  - **`for`** checks whether the condition holds, then iterates once over the code, and finally execute the third expression provided to it, and repeats the same.

```
while (someCondition) {
    // DO SOMETHING
}
```

```
do {
    // DO SOMETHING
} while (someCondition);
```

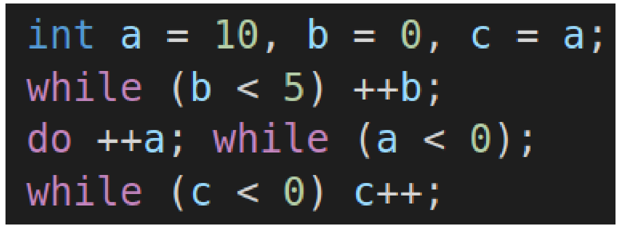
```
/* for (variable initialization;
        condition on the variable;
        expression on the variable) */
for (unsigned int i = 0; i < SomeNumber; i++) {
    /* code */
}
```

- **`for`** receives three inputs (lines of code), each separated by a "**`;`**". The first input is invoked only once and is used for initialization of the variable which the condition is checked against i.e. the second input. Prior to each iteration first the condition is checked, if it does not hold the loop ends.
- All three inputs to **`for`** loop are optional, **`for(;;)`** loops forever.

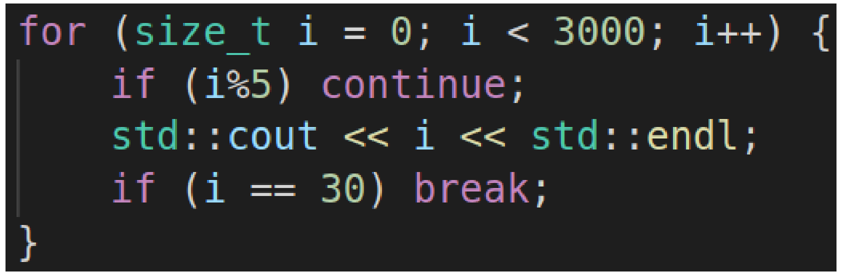


http://www.cplusplus.com/doc/tutorial/control/

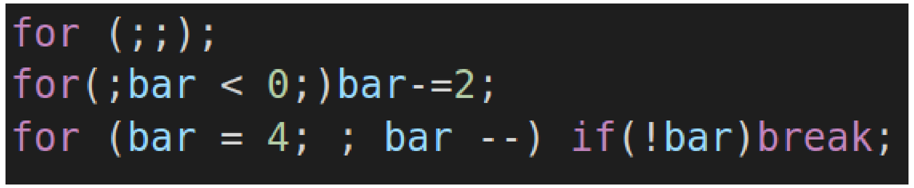
Loops



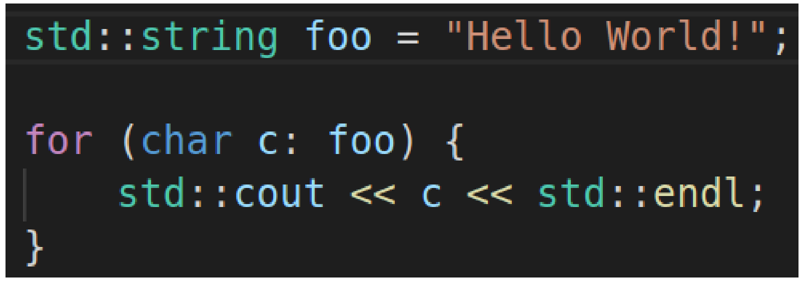
```cpp
int a = 10, b = 0, c = a;
while (b < 5) ++b;
do ++a; while (a < 0);
while (c < 0) c++;
```



```cpp
for (size_t i = 0; i < 3000; i++) {
    if (i%5) continue;
    std::cout << i << std::endl;
    if (i == 30) break;
}
```



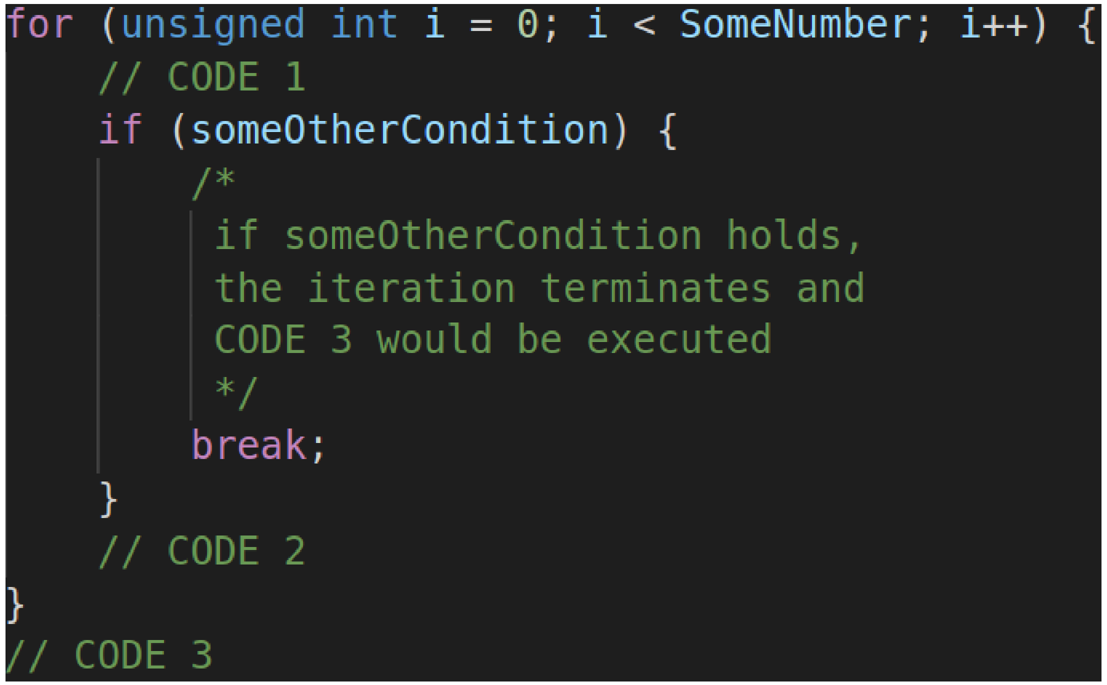
```cpp
for (;;);
for(;bar < 0;)bar-=2;
for (bar = 4; ; bar --) if(!bar)break;
```



```cpp
std::string foo = "Hello World!";

for (char c: foo) {
    std::cout << c << std::endl;
}
```

Control Flows: Jump Statements

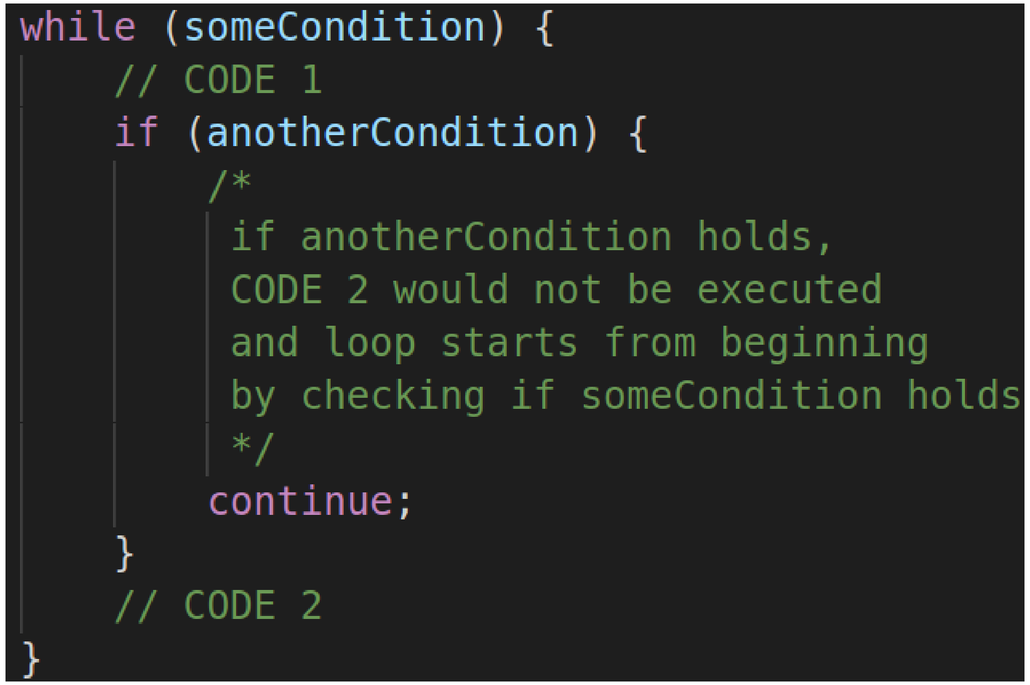
- They are able to alter the flow of execution:
  - **`break`** jumps out of loop or switch selection as if the loop ends at that line
  - **`continues`** skips the rest of the loops and continues to the next iteration as if that line is the last line of the loop
  - **`goto`** jumps to the *"label"* provided to it. An identifier followed by a colon ":" is called a *label.*



```
for (unsigned int i = 0; i < SomeNumber; i++) {
    // CODE 1
    if (someOtherCondition) {
        /*
         if someOtherCondition holds,
         the iteration terminates and
         CODE 3 would be executed
         */
        break;
    }
    // CODE 2
}
// CODE 3
```

Control Flows: Jump Statements

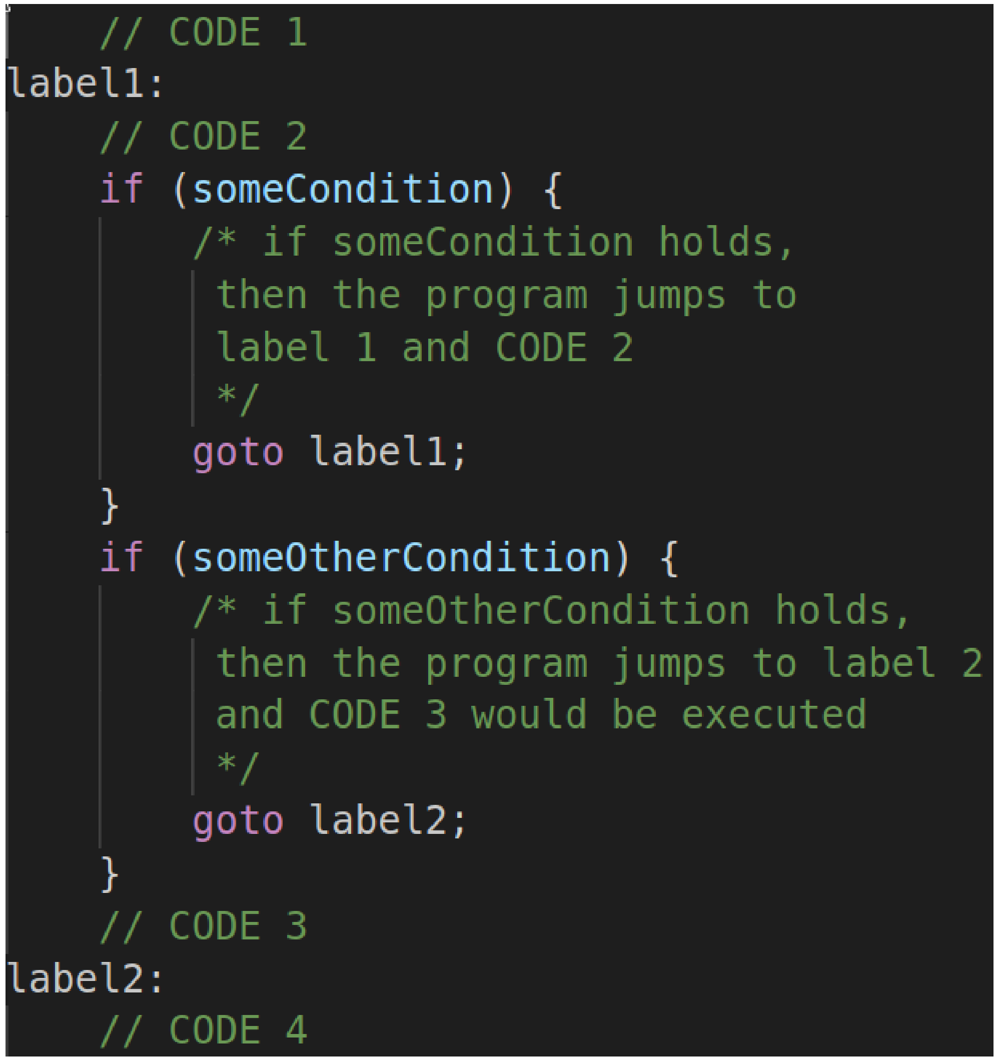
- They are able to alter the flow of execution:
  - **`break`** jumps out of loop or switch selection as if the loop ends at that line
  - **`continue`** skips the rest of the loops and continues to the next iteration as if that line is the last line of the loop
  - **`goto`** jumps to the *"label"* provided to it. An identifier followed by a colon ":" is called a *label.*



```
while (someCondition) {
    // CODE 1
    if (anotherCondition) {
        /*
         if anotherCondition holds,
         CODE 2 would not be executed
         and loop starts from beginning
         by checking if someCondition holds
         */
        continue;
    }
    // CODE 2
}
```

Control Flows: Jump Statements

- They are able to alter the flow of execution:
  - **`break`** jumps out of loop or switch selection as if the loop ends at that line
  - **`continues`** skips the rest of the loops and continues to the next iteration as if that line is the last line of the loop
  - **`goto`** jumps to the "*label*" provided to it. An identifier followed by a colon ":" is called a *label.*



```
    // CODE 1
label1:
    // CODE 2
    if (someCondition) {
        /* if someCondition holds,
         then the program jumps to
         label 1 and CODE 2
         */
        goto label1;
    }
    if (someOtherCondition) {
        /* if someOtherCondition holds,
         then the program jumps to label 2
         and CODE 3 would be executed
         */
        goto label2;
    }
    // CODE 3
label2:
    // CODE 4
```

- **`goto`**s are inherited from C and are not considered a good practice: https://stackoverflow.com/questions/3517726/what-is-wrong-with-using-goto
- An identifier followed by a colon ":" is called a *label.*

Functions

- Functions are used to structure the code.

- Basic function declaration is as follows:

```
ReturnDatatype FunName(InputDatatype Input1_Name …){
    // FUNCTION BODY
}
```

- **void** is used as a return datatype for functions without a return value.
- **main** if used as function name, define the entry point of the program i.e. is the first (only) function being invoked when program executed. If a branch of code is not accessible from the body of the main function, it will not be invoked throughout the execution.
- **size_t** is the same as **unsigned long int** -- it is the *defacto* type used for size. The definition is declared using **#define** preprocessing directive.

```
void printChar(char _c) {
    std::cout << "The charachter is: " << _c << std::endl;
}
```

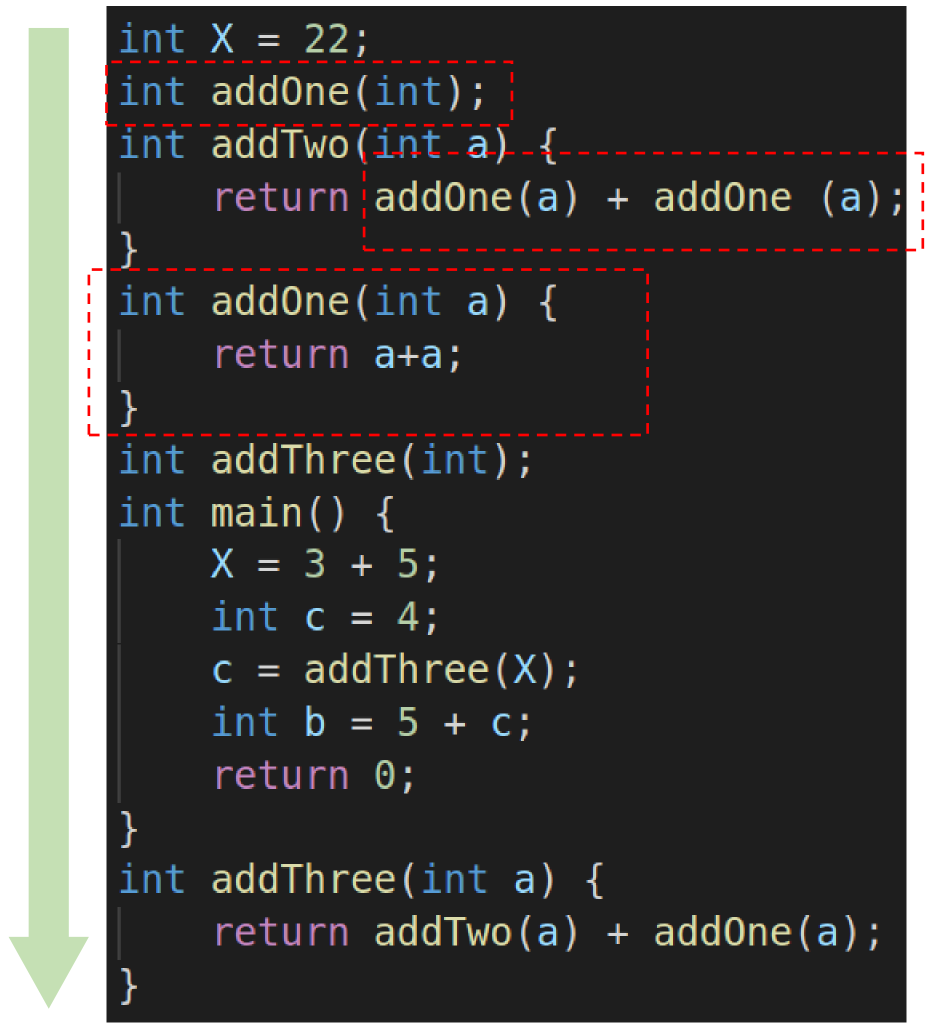
```
char char_a = 'a';
printChar(char_a);
printChar('a');
```

```
long int calcArraysTotalSum(int _array[], size_t _size) {
    long int sum = 0;
    for (size_t i = 0; i < _size; i ++) {
        sum += _array[i];
    }
    return sum;
}
```

```
int arrayOfIntegers[] = {100,200,300};
long int sum = calcArraysTotalSum(arrayOfIntegers,3);
```

Order

- Visibility order is downwards.
- Forward declaration could be used.



```
int X = 22;
int addOne(int);
int addTwo(int a) {
    return addOne(a) + addOne (a);
}
int addOne(int a) {
    return a+a;
}
int addThree(int);
int main() {
    X = 3 + 5;
    int c = 4;
    c = addThree(X);
    int b = 5 + c;
    return 0;
}
int addThree(int a) {
    return addTwo(a) + addOne(a);
}
```

Functions

```cpp
int f3(int foo, int bar) {
    return foo + bar;
}

void f2(int foo = 1) {
    std::cout << "Foo is: " << foo << std::endl;
}
```

```cpp
f2(f3(f3(0,3),3));

if (f3(0,0)) f2(11);
else         f2();
```

```cpp
int fact(int n = 1) {
    int ret = 1;
    std::cout << ">>> Getting into the function wiht n: " << n  << std::endl;
    if (n > 1)
        ret = n * fact(n-1);
    std::cout << "<<< Getting out of the function wiht n: " << n << " and ret: "<< ret << std::endl;
    return ret;
}
```

Exercises !

- Write a program that takes three integers, and prints out the smallest number.
- Write a program that reads a positive integer and checks if it is a perfect square.
- Write a program to make a simple calculator using switch-case. The calculator takes the operation
- (+ or – or * or /) and takes the two input arguments and print the results.
- Write a program that reads a positive integer and computes the factorial.
- Write a program that reads a positive integer and checks if it is a prime.
- Write a program to reverse a number.
- Write a program to count number of digits in a decimal number.

Assignment 1

- Write a program that use the bubble sort algorithm to sort an integer array in ascending order (search for the bubble sorting algorithm).
- Write a program that use the selection sort algorithm to sort an integer array in ascending order (search for the selection sorting algorithm).
- Write a program to return an array containing the values between two 8-bits unsigned integers IN DESCENDING ORDER EXCLUSIVE. The function takes 2 values (LowerValue and UpperValue), it shall determine the values in between, and then arrange the sequence in descending order excluding the upper and lower values.

If the LowerValue is greater than or equal the UpperValue, return an array of 2 elements, both containing value = 0xFF

Example:

Input: LowerValue=2 and UpperValue=5 Output:

Output Array=4,3 Output Array Size=2

Bonus

- Write a Function that calculate the Fibonacci series using recursive method. The Fibonacci Series : 0,1,1,2,3,5,8,13,21,…

- You are designing a poster which prints out numbers with a unique style applied to each of them. The styling is based on the number of closed paths or holes present in a giver number. The number of holes that each of the digits from 0 to 9 have are equal to the number of closed paths in the digit. Their values are:

  1, 2, 3, 5 and 7 = 0 holes.

  0, 4, 6, and 9 = 1 hole.

  8 = 2 holes.

  Given a number, you must determine the sum of the number of holes for all of its digits. For example, the number 819 has 3 holes.

  Function Description

  Complete the function countHoles. The function must return an integer denoting the total number of holes in num.