Akademin

C++ Language

Introduction

C++ Programming Language

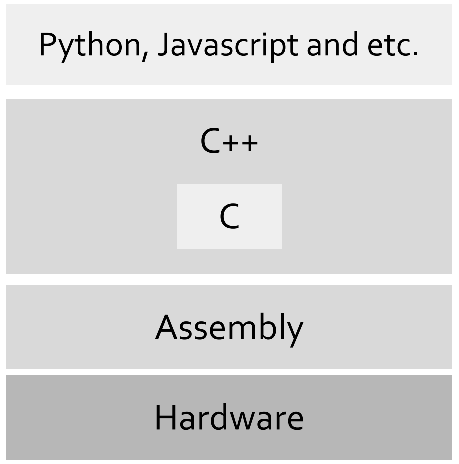
- **C++** is a general purpose programming language

🔗 History of C++

  - Developed by Bjarne Stroustrup at Bell Labs over a period starting in 1979.
  - Earlier it was called as “C with Classes” and it was named C++ in 1983.
  - ANSI (American National Standards Institute) C++ committee founded in 1990
  - ISO (International Organization for Standardization) C++ committee founded in 1991
  - In 1998, ISO published the first C++ standard as ISO/IEC 14882:1998.
    - Known as **C++98**
  - There are different versions; C++98, C++03, C++11, **C++14**, **C++17**, C++20
  - Is used in different high performance applications like
    - Embedded and real-time systems, game programming, simulators, and etc.

C++ Programming Language

- Is a high level programming language with low level capabilities
- Is an efficient machine independent and platform dependent language
- C++ supports
  - Functional and modular programming like C
    - Almost all the features of C are supported.
      - For example, variable length array is not supported
    - Can be used as a better C
      - C libraries can be used. E.g. cstdio, cmath, cstdlib, and ect.
  - Object oriented and generic programming
- C++ is a powerful, complex and unsafe language. E.g. [unsafe memory management]
- Right now (September 2024) [C++ is the second most popular language]



Akademin

C++ Programming Language

- A C++ program can be organized in libraries, modules and components
  - C++ provides a standard library. E.g. cstdio, iostream and etc.
- A C++ program is organized in header and source files.
  - A header file is used to declare types, functions, macros and etc.
  - A source file is used to implement functionalities
  - Different extensions are used for header and source files
    - But as a convention **.h** is used for header files and **.cpp** is used for source files.
- Every program shall implement **main** function as the entry point
- There are different Compilers (gcc, clang, etc.) and build systems (make, cmake, etc.) for C++
- A C++ program can be compiled using g++ or gcc (libstdc++ shall be linked, i.e. -lstdc++)

The First C++ Program

- ***#include*** is a preprocessor directive to tell to the compiler to include header files to the program
- ***main*** function is the entry point to the program. main function can be defined in different ways
  - `int main(void) { /* ... */ }`
  - `int main(int argc, char *argv[]) { /* ... */ }`
  - `int main(int argc, char **argv) { /* ... */ }`
- ***printf*** and **std::cout** are used to print to the standard output(terminal). printf is a C function defined in cstdio and cout is a C++ object defined in iostream
- In C++ we can comment codes in 2 ways:
  - ***Block comment*** which starts with /* and ends with */
  - ***Line comment*** which starts with // and ends with the next newline

```
#include <cstdio> // Inserts content of cstdio into the file
#include <iostream> // Inserts content of iostream into the file

// Definition of the main function
int main(void) {
   printf("Hello ");  /* Print Hello to the standard output */
   std::cout << "World!\n"; /* Print World! to the standard output */

   /* An integer shall be returned to the OS. 0 means that the
     program is completed successfully. A non-zero integer
     means that the program is terminated abnormally. In the case
     of some error usually 1 is returned. */
   return 0;
}
```

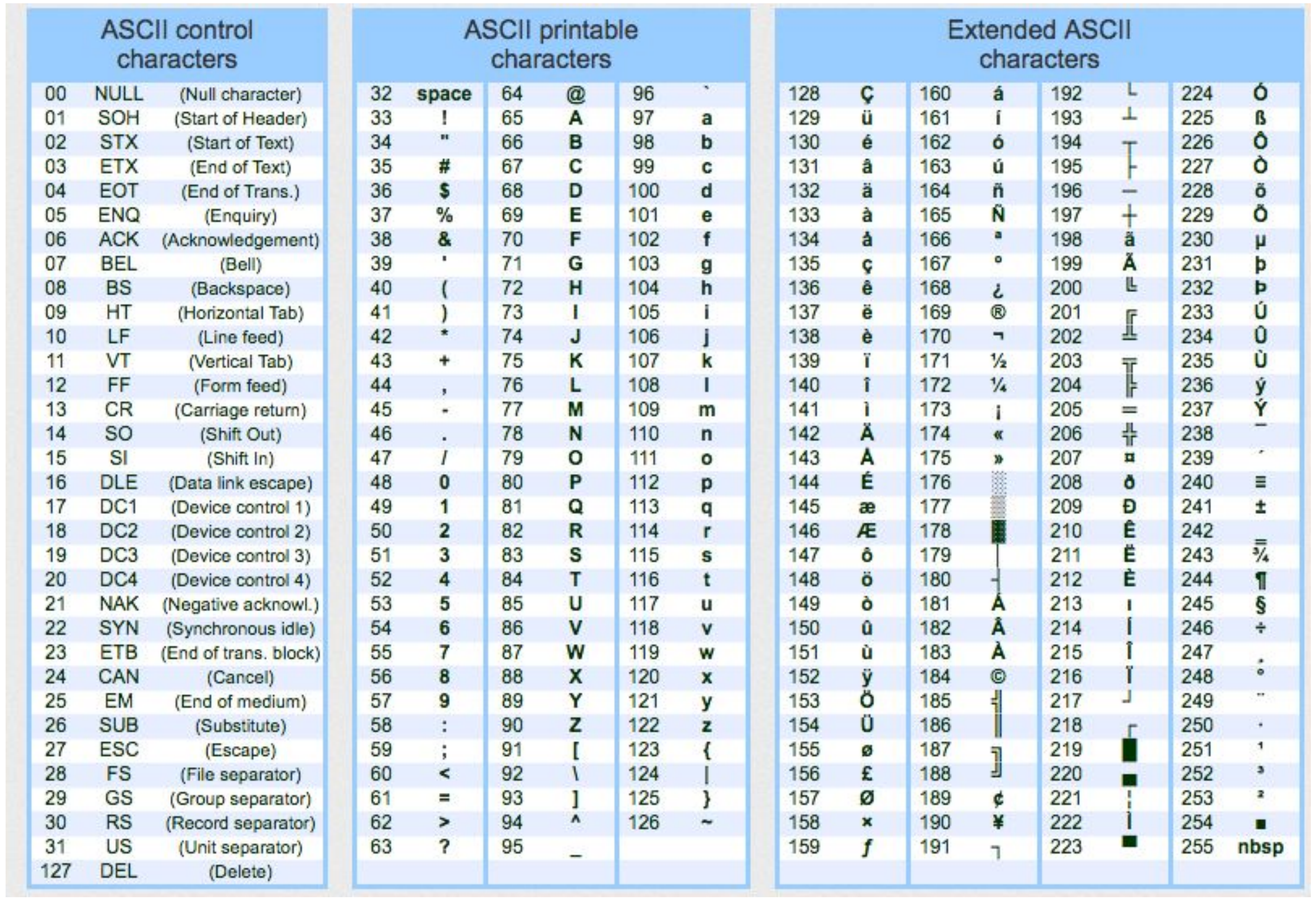
Character Set

- ❖ C++ like a natural language has a set of characters, syntax and semantics
- ❖ The character set of C++ is grouped in two categories
  - ➢ Source character set which contains printable characters
    - ■ **Alphabets** which are **a-z** and **A-Z**
    - ■ **Digits** which are **0, 1, 2, 3, 4, 5, 6, 7, 8, 9**
    - ■ **Special Characters** which are **! " # % & ' () * + , − . / : ; < = > ? [\ ] ^ _ { | } ~**.
    - ■ **White Spaces** which are Space( ), horizontal tab(**\t**), vertical tab(**\v**), newline(**\n**), and form feed(**\f**)
  - ➢ Execution character set
    - ■ Contains non-printable characters that performs some functionalities during execution
    - ■ E.g. null(**\0**), alert(**\a**), backspace(**\b**), carriage return(**\r**) and etc.
- ● All the characters in the character set of C++ are defined in the ASCII table

American Standard Code for Information Interchange (ASCII)

- **ASCII** is a character encoding standard for electronic communication. It is a 7-bit encoding system and each character in ASCII is assigned a number between 0 and 127

- **Extended ASCII** is an 8-bit character encoding that includes the standard seven-bit ASCII characters, plus additional characters.



ASCII control characters			ASCII printable characters						Extended ASCII characters							
00	NULL	(Null character)	32	space	64	@	96	`	128	Ç	160	á	192	└	224	Ó
01	SOH	(Start of Header)	33	!	65	A	97	a	129	ü	161	í	193	┴	225	ß
02	STX	(Start of Text)	34	"	66	B	98	b	130	é	162	ó	194	┬	226	Ô
03	ETX	(End of Text)	35	#	67	C	99	c	131	â	163	ú	195	├	227	Ò
04	EOT	(End of Trans.)	36	$	68	D	100	d	132	ä	164	ñ	196	─	228	õ
05	ENQ	(Enquiry)	37	%	69	E	101	e	133	à	165	Ñ	197	┼	229	Õ
06	ACK	(Acknowledgement)	38	&	70	F	102	f	134	å	166	ª	198	ã	230	µ
07	BEL	(Bell)	39	'	71	G	103	g	135	ç	167	º	199	Ã	231	þ
08	BS	(Backspace)	40	(	72	H	104	h	136	ê	168	¿	200	╚	232	Þ
09	HT	(Horizontal Tab)	41	)	73	I	105	i	137	ë	169	®	201	╔	233	Ú
10	LF	(Line feed)	42	*	74	J	106	j	138	è	170	¬	202	╩	234	Û
11	VT	(Vertical Tab)	43	+	75	K	107	k	139	ï	171	½	203	╦	235	Ù
12	FF	(Form feed)	44	,	76	L	108	l	140	î	172	¼	204	╠	236	ý
13	CR	(Carriage return)	45	-	77	M	109	m	141	ì	173	¡	205	═	237	Ý
14	SO	(Shift Out)	46	.	78	N	110	n	142	Ä	174	«	206	╬	238	¯
15	SI	(Shift In)	47	/	79	O	111	o	143	Å	175	»	207	¤	239	´
16	DLE	(Data link escape)	48	0	80	P	112	p	144	É	176	░	208	ð	240	≡
17	DC1	(Device control 1)	49	1	81	Q	113	q	145	æ	177	▒	209	Ð	241	±
18	DC2	(Device control 2)	50	2	82	R	114	r	146	Æ	178	▓	210	Ê	242	‗
19	DC3	(Device control 3)	51	3	83	S	115	s	147	ô	179	│	211	Ë	243	¾
20	DC4	(Device control 4)	52	4	84	T	116	t	148	ö	180	┤	212	È	244	¶
21	NAK	(Negative acknowl.)	53	5	85	U	117	u	149	ò	181	Á	213	ı	245	§
22	SYN	(Synchronous idle)	54	6	86	V	118	v	150	û	182	Â	214	Í	246	÷
23	ETB	(End of trans. block)	55	7	87	W	119	w	151	ù	183	À	215	Î	247	¸
24	CAN	(Cancel)	56	8	88	X	120	x	152	ÿ	184	©	216	Ï	248	°
25	EM	(End of medium)	57	9	89	Y	121	y	153	Ö	185	╣	217	┘	249	¨
26	SUB	(Substitute)	58	:	90	Z	122	z	154	Ü	186	║	218	┌	250	·
27	ESC	(Escape)	59	;	91	[	123	{	155	ø	187	╗	219	█	251	¹
28	FS	(File separator)	60	<	92	\	124	|	156	£	188	╝	220	▄	252	³
29	GS	(Group separator)	61	=	93	]	125	}	157	Ø	189	¢	221	¦	253	²
30	RS	(Record separator)	62	>	94	^	126	~	158	×	190	¥	222	Ì	254	■
31	US	(Unit separator)	63	?	95	_			159	ƒ	191	┐	223	▀	255	nbsp
127	DEL	(Delete)														

Akademin

C++ Identifiers

- Identifiers: Names of variables, functions, and other elements used in a C++ program
- Identifiers can contain letters (**a-z** and **A-Z**), digits (**0-9**) and underscores (**_**)
- The first character of an identifier can not be a digit.
- Identifiers are case-sensitive and there is no limit on the length of an identifier.
  - Some compilers have limitations. E.g. Microsoft C++: 2048 characters
- Identifiers that contain a **double underscore** or begin with **an underscore followed by an uppercase letter** are reserved by the implementation and shall not be used.
  - For example: **__x**, **_Max** and **__LINE__**
- Identifiers begin with an underscore are reserved by the implementation for use as a name in the global namespace. E.g. **_a12** can not be used as a **global** identifier

C++ Identifiers

❖ The reserved keywords in C++ must not be used as identifiers.

C++ reserved keywords							
alignas	alignof	and	and_eq	asm	auto	bitand	bitor
bool	break	case	catch	char	char16_t	char32_t	class
compl	concept	const	const_cast	constexpr	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	requires	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		

Identifier Scope

- Scope of an identifier refers to the part of a code in which the identifier is accessible
- The scope of an identifier is determined by the location the identifier is declared
- Generally scope of an identifier begins after its declaration.
    - There are some exceptions; e.g. labels, tag names and enumeration constants
- In C++ there are five types of scope
    - Global scope: An identifier not declared inside a language construct (e.g., a class or a function).
    - Namespace scope: An identifier declared in a namespace and not inside some language construct (e.g., a class or a function). The global scope is a namespace scope
    - Local scope: An identifier declared in function parameters or in a function body
    - Class scope: An identifier which is a member of a class.
    - Statement scope: An identifier declared in a for, if, or switch statement.

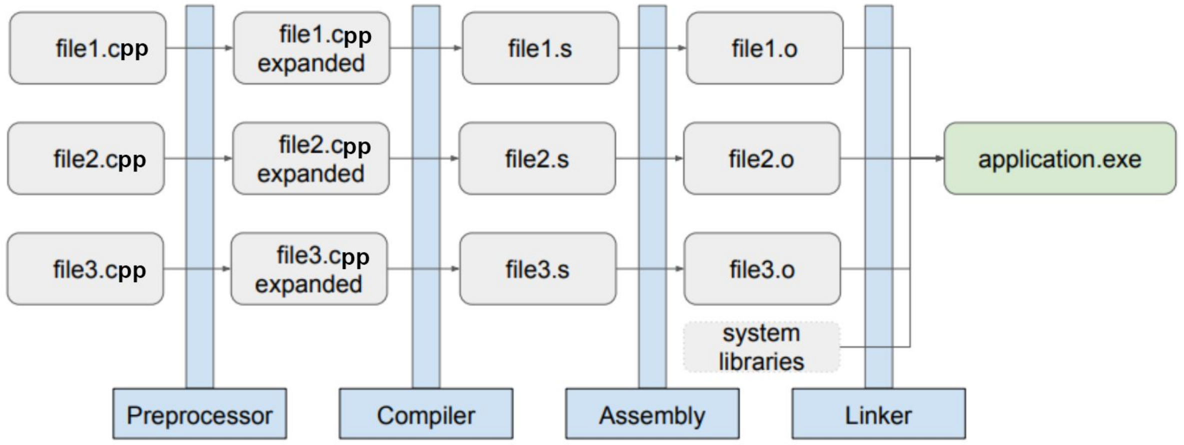
Visual Studio Code Settings

- We use vscode to write C++ code.
- Install the C/C++ (ms-vscode.cpptools) extension in vscode
- Enable code formatting in vscode
  - Click on File > Preferences > Settings in the main menu
  - Click on Text Editor > Formatting and enable Format on **Past**, **Save**, **Save Mode** and **Type**
  - Click on **Extensions > C / C++**, find **C_Cpp: Clang_format_fallback Style** and use Visual Studio
- **Doxygen** is the de facto standard tool for generating documentation from source files
- Install **Doxygen Documentation Generator** extension in Visual Studio Code
  - In the setting click on **Extensions** and find **Doxygen Documentation Generator**
  - Scroll down and find **Generic: Author Email** and **Generic: Author Name** and fill them with your name and your email address

Build Process

- Build process of a C++ program using g++
  - **Preprocessor**: ‘#’-prefixed lines for includes, replacing macros, conditional compilation, et.c.
  - **Compiler**: Generate assembly code from the preprocessed code, checks the code for errors
  - **Assembler**: Makes machine instructions (object file) from the generated assembly code
  - **Linker**: Resolves symbols (function calls, global variables...) between software components and system libraries

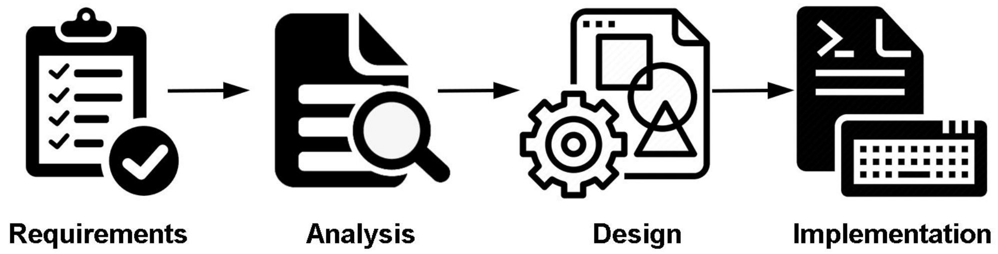
- *g++ -E file.cpp -o file.ii => Preprocessed code*
- *g++ -S file.cpp -o file.s => Assembly code*
- *g++ -c file.cpp -o file.o=> Object file*
- *g++ file1.o file2.o file3.o -o app => Linked file(app)*
- *g++ main.cpp -save-temps -o main*
- *g++ file1.cpp file2.cpp file3.cpp -o application*



Akademin

Basic Development Model

- Requirements
  - The requirement specification according to the customer needs
- Analysis
  - Analyzing the requirements in order to specify exactly what the software attempts to do
  - How we can completely fulfill the requirements
- Design
  - Design the software using well-trusted tools, methods and techniques according to the analysis in the previous phase
- Software implementation
  - Coding and development cycle based on the design phase.



Akademin