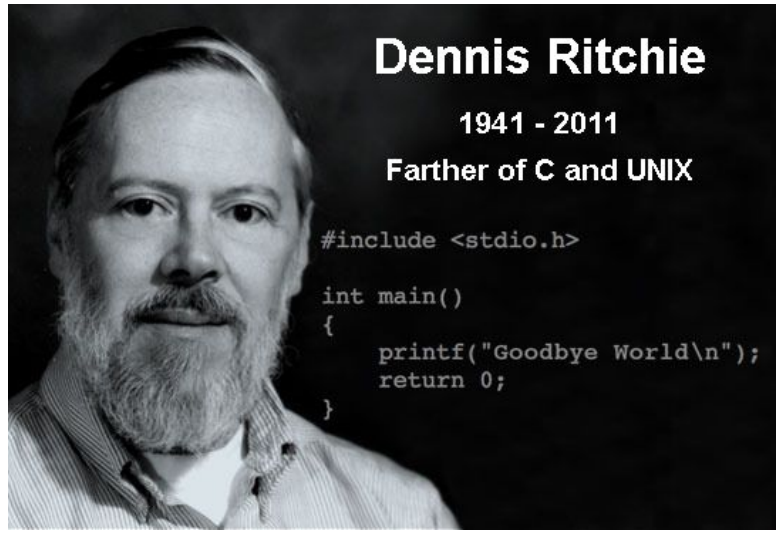
C Programming

Introduction

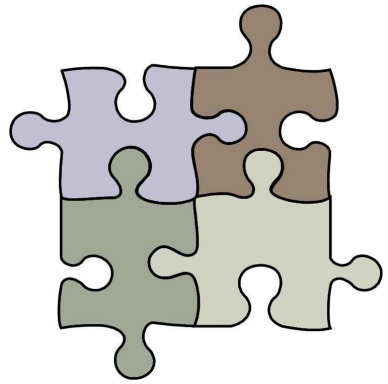
C Programming Language

- **C** is a general purpose, procedural programming language
  - Developed by Dennis Ritchie in 1972 at Bell Labs
  - In 1978, Brian Kernighan and Dennis Ritchie published the first edition of The C Programming Language. This version is known as **K&R C**
  - Standardized in 1989 by the American National Standards Institute (ANSI) - **ANSI C**
  - In 1990, the ANSI C standard was adopted by the International Organization for Standardization (ISO) as ISO/IEC 9899:1990. Known as **C90**
  - There are different versions; C90, C99, **C11**, C17
  - We focus on C11
  - Is used in different areas of computer programming



C Programming Language

- C is a high level programming language with low level capabilities
- C is an efficient machine independent and platform dependent language
- C is a functional programming language and capable of modular programming
  - Functions are the procedural building blocks of C programs
    - Every program shall define **main**() which is the entry point to the program
  - A C program can be organized in libraries, modules and components
    - C has some standard libraries. E.g. stdio, math, stdlib and etc.



- C is known as the mother of other languages
  - Languages like C++, Java, JavaScript, Perl, PHP and etc. have been influenced by C
- Right now (september 2021) C is the most popular language
- C is a small and an easy to learn language. *No native string, no garbage collector and etc.*

What is a C program?

- Header (**.h**) files which contain
  - Includes to include libraries
  - Function declarations
  - Variables and constants
  - And etc.
- Source code (**.c**) files which contain
  - Includes to include libraries
  - Local function declarations & definitions
  - Local variables
  - And etc.
- Compilers (GNU **gcc**, clang, msvc etc.)
- Build systems (GNU **make**, cmake etc.)

```
#ifndef _MY_HEADER_H_                           // Include guards
#define _MY_HEADER_H_

/* ------------------------------ Includes ------------------------------ */
#include <stdio.h>                              //Standard includes has angle brackets around
#include <stdbool.h>

/* ------------------------- Constants and Types ------------------------- */
#define TIMES_TO_SAY                    10      //Defines are exchanged during compilation
static const char* textToSay= "Hello world!\n";

/* --------------------- Exported Function Prototypes --------------------- */
bool DoTheThing(int timesToSay);

#endif //_MY_HEADER_H_
```

An example header file

```
/* ------------------------------ Includes ------------------------------ */
#include <stdio.h>                              //Standard includes has angle brackets around
#include <stdbool.h>
#include "MyHeader.h"                           /*Project specific include */

/* ------------------------------ Variables ------------------------------ */
static int times = 0;

/* ---------------------- Local Function Prototypes ---------------------- */
int decreseByOne(int value);

/* --------------------- Exported Function Prototypes --------------------- */
int main(int argc /*Number of arguments*/, char* argv[] /*Argument array*/)   //Entrypoint of application
{
    bool retVal = DoTheThing(TIMES_TO_SAY);
    retVal ? return 0 : return -1;
}

bool DoTheThing(int timesToSay)
{
    ...

/* -------------------------- Local Functions -------------------------- */
int decreseByOne(int value)
...
```

An example source file

The First C Programming

- ***#include*** is a preprocessor directive which tells the compiler to include stdio library to the program

```c
#include <stdio.h> // Preprocessor directive

int main(void) // Definition of the main function
{
    printf("Hello World!\n"); /* Print Hello World! to the terminal */

    /*
    return 0 implies that the program ran without an error and succesfully exited.
    return 1 implies that the program had an error and did not successfully run.
    Usually return 1 is used to denote that the program did not run as expected.
    */
    return 0;
}
```

- ***main*** function is the entry point to the program. Every C program shall define this function. main function can be defined in different ways
  - `int main(void) { /* ... */ }`
  - `int main(int argc, char *argv[]) { /* ... */ }`
  - `int main(int argc, char **argv) { /* ... */ }`
- ***printf*** is a function in stdio.h which is used to print a formatted string to the standard output (terminal)
- In C we can comment codes in 2 ways:
  - ***Block comment*** which starts with */** and end with ***/***
  - ***Line comment*** which starts with *//* and end with the next newline

Character Set of C

- 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**), and carriage return(**\r**)

- 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

00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters

32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Extended ASCII characters

128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

Identifiers in a C Program

- identifiers: Names of variables, functions, and other elements defined in a C program
- Identifiers can contain letters (**a-z** and **A-Z**), digits (**0-9**) and underscore (_)
- The first character of an identifier can not be a digit
- Identifiers are case-sensitive
- There is no limit on the length of identifiers. But
  - At least the first 31 characters of identifiers with external linkage shall be significant
    - E.g. aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa1 and aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa2 are not OK.
  - At least the first 63 characters of identifiers with internal linkage shall be significant
- All identifiers begin with an **underscore** followed by a **second underscore** or an **uppercase letter** are reserved and shall not be used. E.g. **__x**, **_Max**, **__LINE__** and etc.

Identifiers in a C Program

- Identifiers begin with an underscore and not followed by an uppercase letter are reserved as identifiers with file scope.
  - For example **_a12** can not be used as the name of a **function** or a **global variable**
- Identifiers defined in any header file you include are reserved.
- The reserved keywords in C must not be used as identifiers.

44 reserved keywords of C										
auto	break	case	char	const	continue	default	do	double	else	enum
extern	float	for	goto	if	inline	int	long	register	restrict	return
short	signed	sizeof	static	struct	switch	typedef	union	unsigned	void	volatile
while	_Alignas	_Alignof	_Atomic	_Bool	_Complex	_Generic	_Imaginary	_Noreturn	_Static_assert	_Thread_local

Identifier Scope

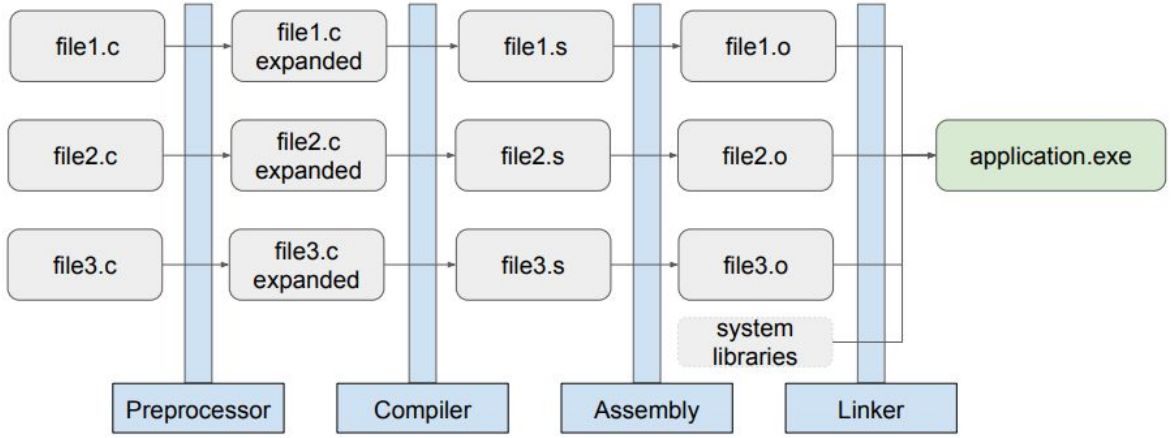
- Scope of an identifier refers to the part of a code in which the identifier is accessible
- The scope of an identifier is determined by the location the identifier is declared
- The scope of an identifier generally begins after its declaration.
  - Exceptions: Tag names of structures, unions, enums, and constants in enums
    - Their scope begin after their appearance in type specifiers that declare the tags
- In C there are four types of scope
  - Block scope: Identifiers declared within a block.
    - A block of code starts with **{** and ends with **}**. Such identifiers are valid only in the block
    - The parameter names in the head of a **function definition** have block scope
    - **Exception**: Labels have function block scope
  - Function prototype scope: The parameter names in a function prototype
  - Function scope: Identifiers declared in the body of functions
  - File scope: Identifiers declared outside of all blocks and parameter list of functions

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 gcc/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



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

Basic Development Phases

- 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.