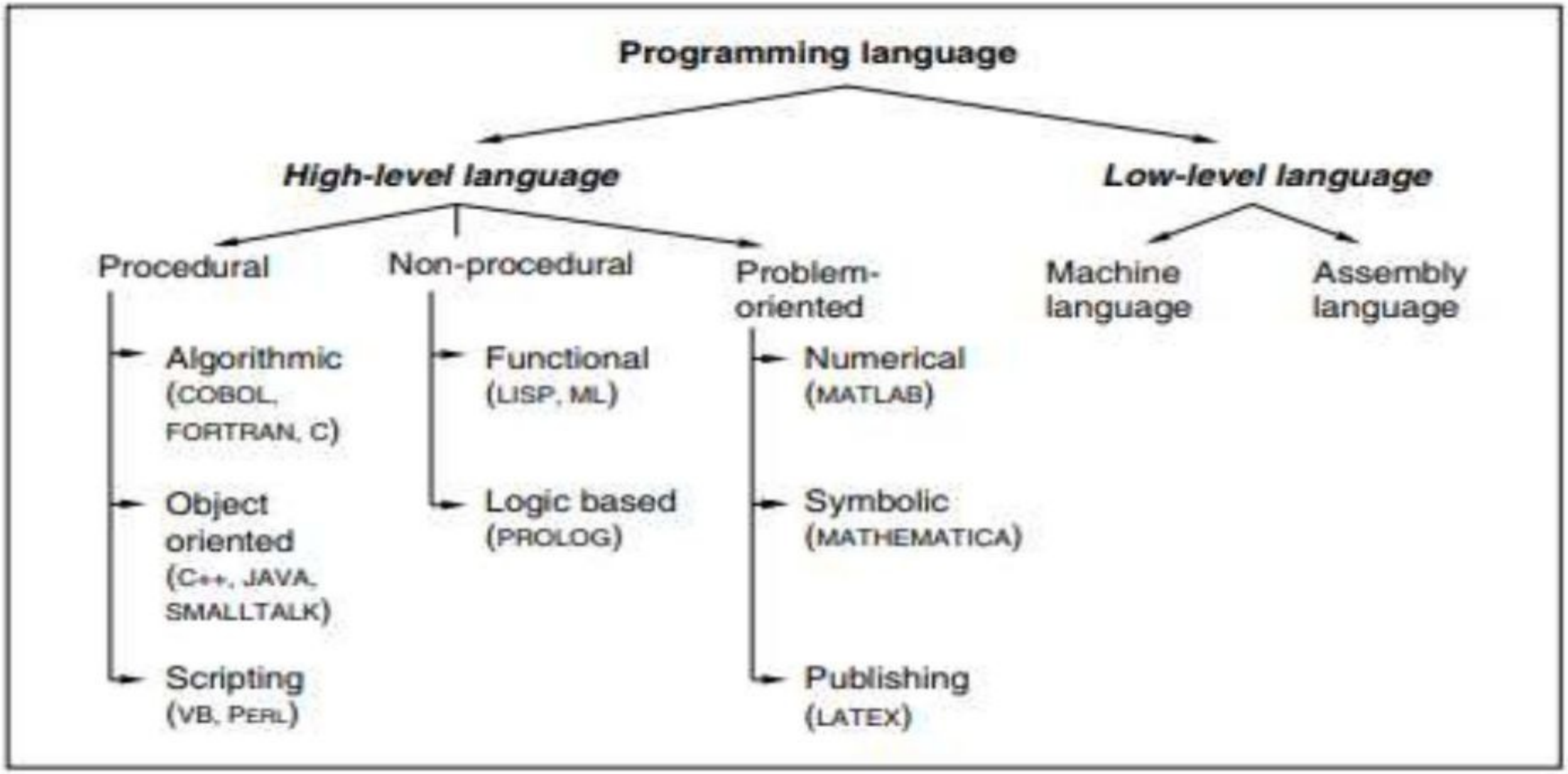
UNIT II

Introduction To C Programming

Introduction To C

Programming Paradigms:



Programming Paradigms:

1. Low-level languages: Assembly language

each assembly language instruction accomplishes only a single operation and the coding for a problem is at the individual instruction level.

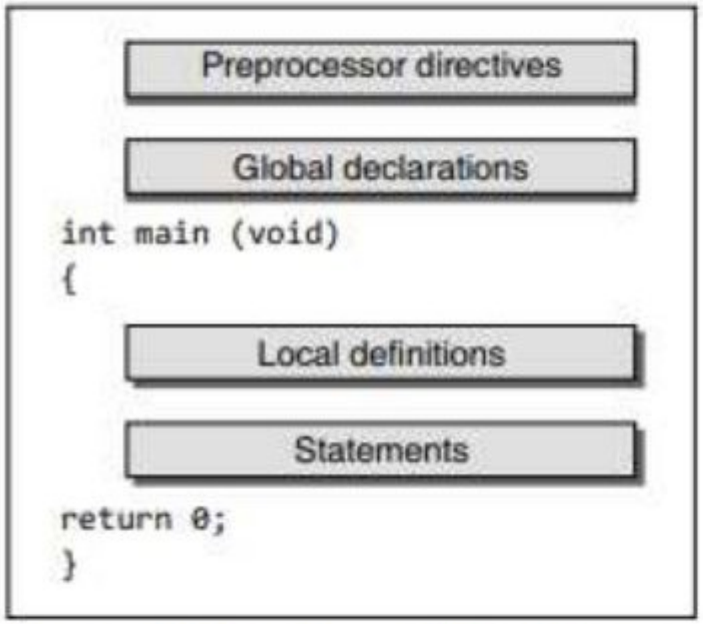
2.The high levelprogramming languagesmay also be categorized into threegroups—procedural, non-procedural, and problem oriented.

□ Procedural programming languages- Algorithmic, Object oriented and Scripting

□ Non Procedural Languages: Rule based languages or logic programming languages

□ Problem-oriented languages: MATLAB is a very popular language among scientists and engineers

Basic Structure of C Program:



```
int main (void)
{
return 0;
}
```

Figure 2. Structure of C program.

The process of compiling and running the C Program:

There are mainly three steps in developing a program in C:

1. Writing the C program
2. Compiling the program
3. Executing the program.

The process of compiling and running the C Program:

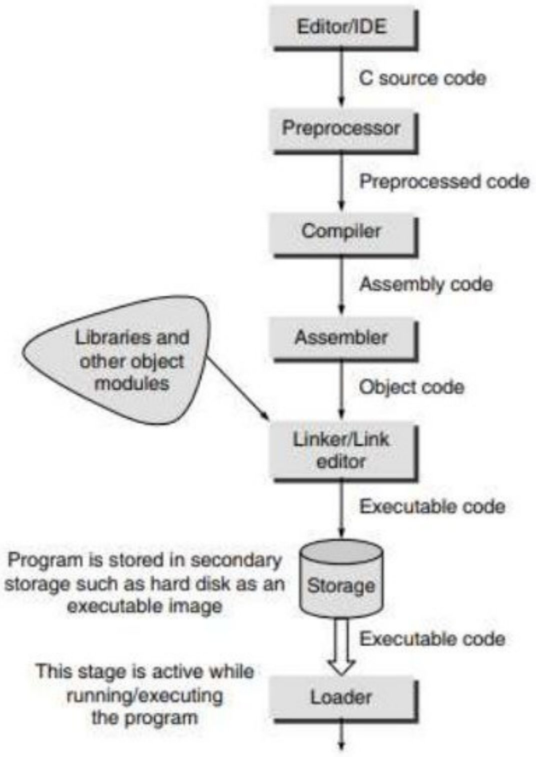

Editor/IDE
C source code
Preprocessor
Preprocessed code
Compiler
Assembly code
Assembler
Libraries and other object modules
Object code
Linker/Link editor
Executable code
Program is stored in secondary storage such as hard disk as an executable image
Storage
Executable code
This stage is active while running/executing the program
Loader

Figure 3. Typical steps for entering, compiling and executing C programs

2.5 Features of C Programming Language:

- ✓ Procedural Language
- ✓ Fast and Efficient
- ✓ Modularity
- ✓ Statically Type
- ✓ General Purpose Language
- ✓ Rich set of built in Operators
- ✓ Libraries with rich Functions
- ✓ Middle Level Language
- ✓ Portability
- ✓ Easy to Extend

Character Set:

character:- It denotes any alphabet, digit or special symbol used to represent information. The characters in C are grouped into the following two categories:

1. Source character set
2. Alphabets
3. Digits
4. Special Characters
5. White Spaces

Character Set:

2. Execution character set

~	tilde	%	percent sign	|	vertical bar	@	at symbol
+	plus sign	<	less than				
_	underscore	-	minus sign	>	greater than	^	caret
#	number sign	=	equal to				
&	ampersand	$	dollar sign	/	slash	(	left parenthesis
*	asterisk	\	back slash				
)	right parenthesis	'	apostrophe	:	colon	[	left bracket
"	quotation mark	;	semicolon				
]	right bracket	!	exclamation mark	,	comma	{	left flower brace
?	Question mark	.	dot operator				
}	right flower brace						

Character Set:

ALPHABETS

Uppercase letters	A-Z
Lowercase letters	a-z

DIGITS 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Special characters:

Whitespace characters:

\b	blank space	\t	horizontal tab	\v	vertical tab	\r	carriage return
\f	form feed	\n	new line				

\\	Back slash	\'	Single quote	\"	Double quote	\?	Question mark
\0	Null	\a	Alarm (bell)				

C Token:

Basic building block of a C-Program

1. Identifier It is a sequence of characters invented by the programmer to identify or name a specific object, and the nameis formed by a sequence of letters, digits, and underscores.
2. Keywords These are explicitly reserved words that have a strict meaning as individual tokens to the compiler. They cannot be redefined or used in other contexts. Use of variable names with the same name as any of the keywords will cause a compiler error.
3. 3. Operators These are tokens used to indicate an action to be taken (usually arithmetic operations, logical operations, bit operations, and assignment operations). Operators can be simple operators (a single character token) or compound operators (two or more character tokens).

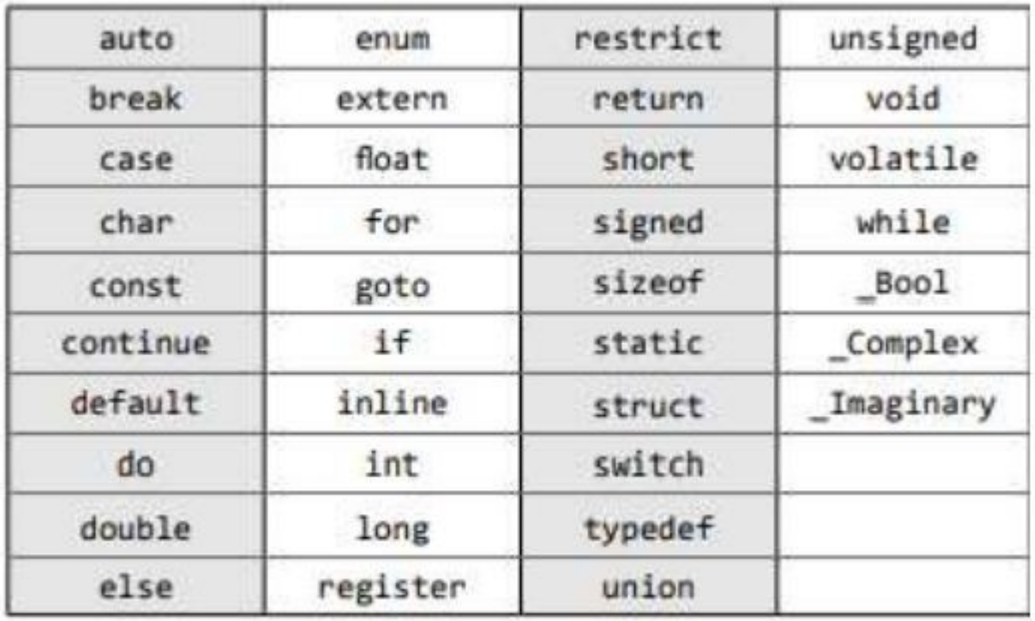
C Token:

Basic building block of a C-Program

4. Separators These are tokens used to separate other tokens. Two common kinds of separators are indicators of an end of an instruction and separators used for grouping.

5. Constant It is an entity that does not change.

Keywords :



auto	enum	restrict	unsigned
break	extern	return	void
case	float	short	volatile
char	for	signed	while
const	goto	sizeof	_Bool
continue	if	static	_Complex
default	inline	struct	_Imaginary
do	int	switch	
double	long	typedef	
else	register	union	

Identifiers :

Identifiers - In C, variables, arrays, functions, and labels are named.

1. The first character must be an alphabetic character (lower-case or capital letters) or an underscore ‘_’.

2. All characters must be alphabetic characters, digits, or underscores.

3. The first 31 characters of the identifier are significant. Identifiers that share the same first 31 characters may be indistinguishable from each other.

4. A keyword cannot be duplicated by an identifier. A keyword is word which has special meaning in C.

Some examples of proper identifiers are employee_number, box_4_weight, monthly_pay, interest_per_annum, job_number, and tool_4. Some examples of incorrect identifiers are 230_item, #pulse_rate, total~amount, /profit margin, and ~cost_per_ item.

Constants:

Fixed values that the program may not alter during its execution.

1. Integer constant: 1, 25, and 23456 are all decimal integer constants, a floating constant
2. Floating Point Constant: 0.002164 may be written in scientific notation as 2.164E-3 or 2.164e-3 The letter E (or e) stands for exponent.
3. Character constantcharacter constant 'A' is equivalent to writing down the hex value 41 or the octal value 101., or a string literal
4. String constant is a sequence of characters enclosed in double quotes. (the null character '\0', to mark the end of the string)

Variables:

- The name of a variable(called as identifier) can be composed of letters, digits, and the underscore character.
- A variable is a name given to a storage area that programs can manipulate.
- Example: a, b, sum, area_tri, pgm_a..

Data Types:

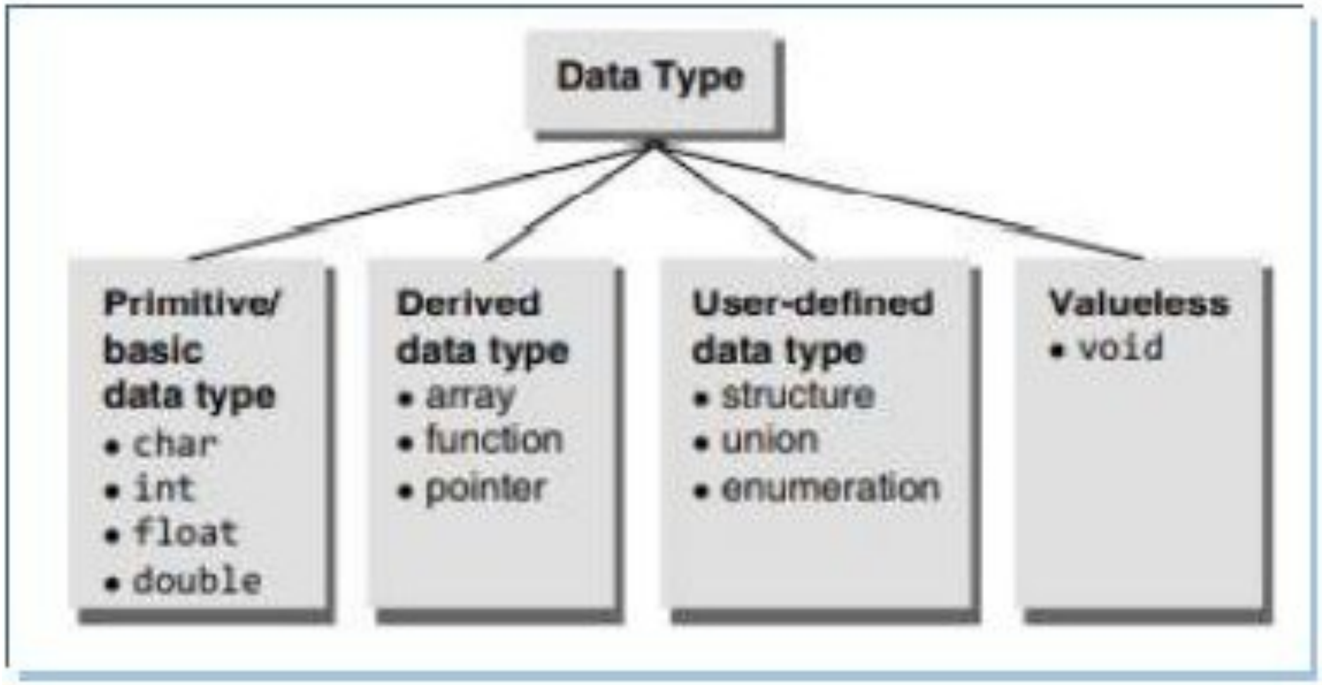
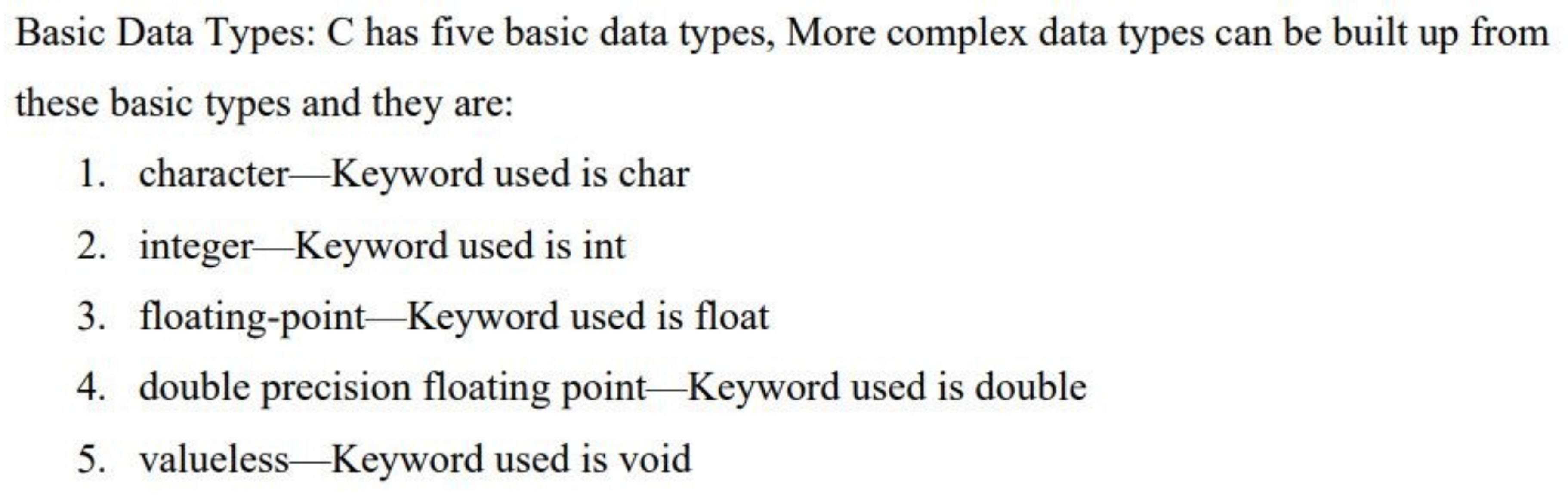


Figure 4. Data types in C

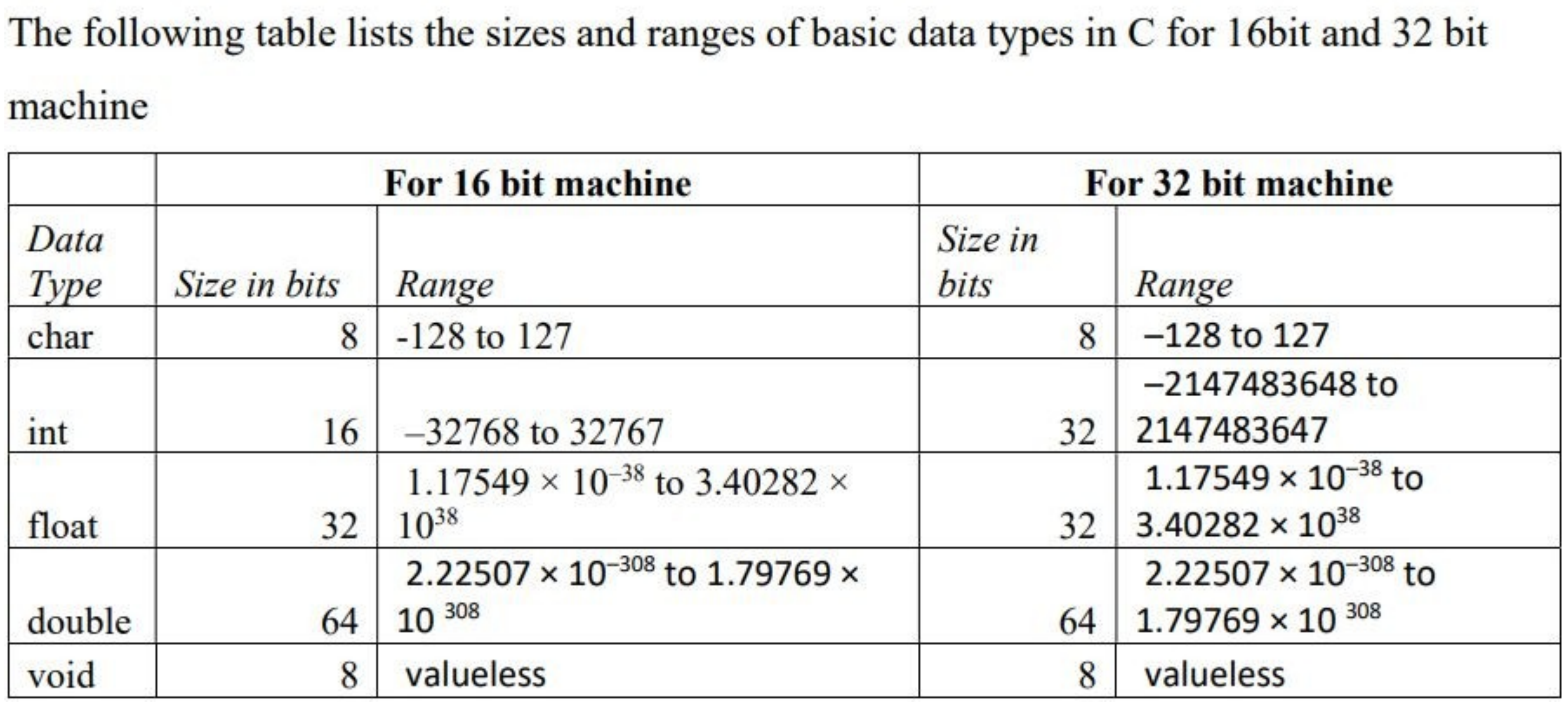
Data Types:



Basic Data Types: C has five basic data types, More complex data types can be built up from these basic types and they are:

1. character—Keyword used is char
2. integer—Keyword used is int
3. floating-point—Keyword used is float
4. double precision floating point—Keyword used is double
5. valueless—Keyword used is void

Data Types:



The following table lists the sizes and ranges of basic data types in C for 16bit and 32 bit machine

	For 16 bit machine		For 32 bit machine	
Data Type	*Size in bits*	*Range*	*Size in bits*	*Range*
char	8	-128 to 127	8	–128 to 127
int	16	–32768 to 32767	32	–2147483648 to 2147483647
float	32	1.17549×10^{-38} to 3.40282×10^{38}	32	1.17549×10^{-38} to 3.40282×10^{38}
double	64	2.22507×10^{-308} to 1.79769×10^{308}	64	2.22507×10^{-308} to 1.79769×10^{308}
void	8	valueless	8	valueless

Pre-Processor Directives

- □ Three uses of the preprocessor are:
  - directives
  - constants
  - macros.
- □ Directives are commands that tell the preprocessor to skip part of a file, include another file, or define a constant or macro.
- □ Directives always begin with a sharp sign - #

Pre-Processor Directives

Header Files

The #include directive tells the preprocessor to grab the text of a file and place it directly into the current file. Typically, such statements are placed at the top of a program--hence the name "header file" for files thus included.

Constants

If we write

#define [identifier name] [value]

whenever [identifier name] shows up in the file, it will be replaced by [value].

If you are defining a constant in terms of a mathematical expression, it is wise to surround the entire value in parentheses:

```
#define PI_PLUS_ONE (3.14 + 1)
```

By doing so, you avoid the possibility that an order of operations issue will destroy the meaning of your constant:

```
x = PI_PLUS_ONE * 5;
```

Pre-Processor Directives

Macros

The other major use of the preprocessor is to define macros. The advantage of a macro is that it can be type-neutral (this can also be a disadvantage, of course), and it's inlined directly into the code, so there isn't any function call overhead. (Note that in C++, it's possible to get around both of these issues with templated functions and the inline keyword.)A macro definition is usually of the following form:

```
#define MACRO_NAME(arg1, arg2, ...) [code to expand to]
```

For instance, a simple increment macro might look like this:

```
#define INCREMENT(x) x++
```

They look a lot like function calls, but they're not so simple. There are actually a couple of tricky points when it comes to working with macros. First, remember that the exact text of the macro argument is "pasted in" to the macro. For instance, if you wrote something like this:

```
#define MULT(x, y) x * y
```

and then wrote

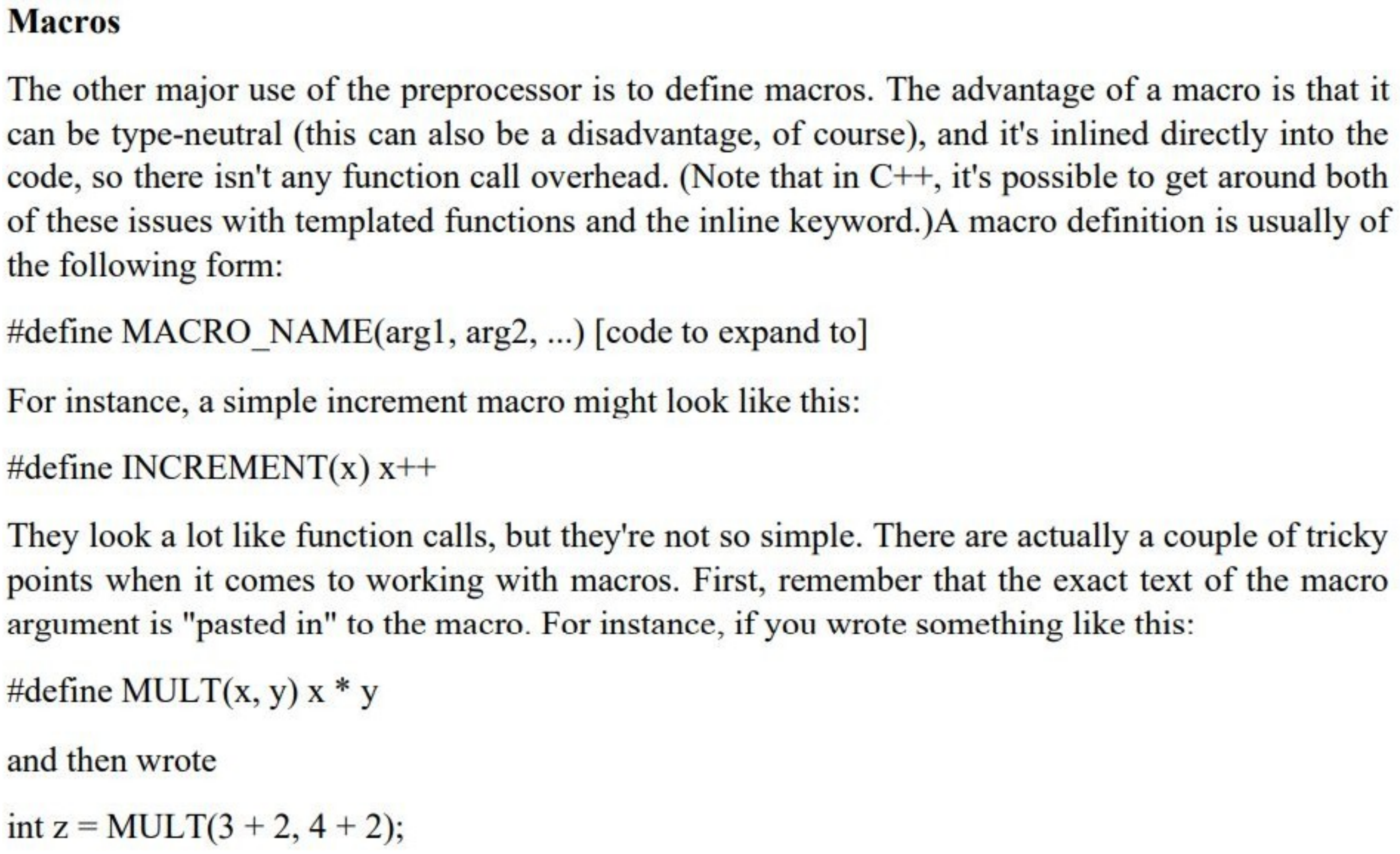
```
int z = MULT(3 + 2, 4 + 2);
```

RV College of Engineering ®

Pre-Processor Directves

Go, change the world

Pre-Processor Directives



Macros

The other major use of the preprocessor is to define macros. The advantage of a macro is that it can be type-neutral (this can also be a disadvantage, of course), and it's inlined directly into the code, so there isn't any function call overhead. (Note that in C++, it's possible to get around both of these issues with templated functions and the inline keyword.)A macro definition is usually of the following form:

```
#define MACRO_NAME(arg1, arg2, ...) [code to expand to]
```

For instance, a simple increment macro might look like this:

```
#define INCREMENT(x) x++
```

They look a lot like function calls, but they're not so simple. There are actually a couple of tricky points when it comes to working with macros. First, remember that the exact text of the macro argument is "pasted in" to the macro. For instance, if you wrote something like this:

```
#define MULT(x, y) x * y
```

and then wrote

```
int z = MULT(3 + 2, 4 + 2);
```

END OF CHAPTER