Unit #2

Fundamentals of C programming

Outline

- Overview of C
  - Structure of C program
  - Data types
  - Keywords
  - Identifiers - Constants and Variables
  - Expressions and operators
- Problem solving with decisions, loops and data structures
  - Decision statements
  - Loop control statements
  - Arrays

OVERVIEW OF C

Background of C

- What is a language?
- Why to use language?
- How Computers perform job?
  - Computer = Hardware + Software
  - Hardware can not do anything alone
  - Software can not imagined without Hardware
- Like human being interact using Hindi, English and other human languages.
  - Computer’s one of the language is C

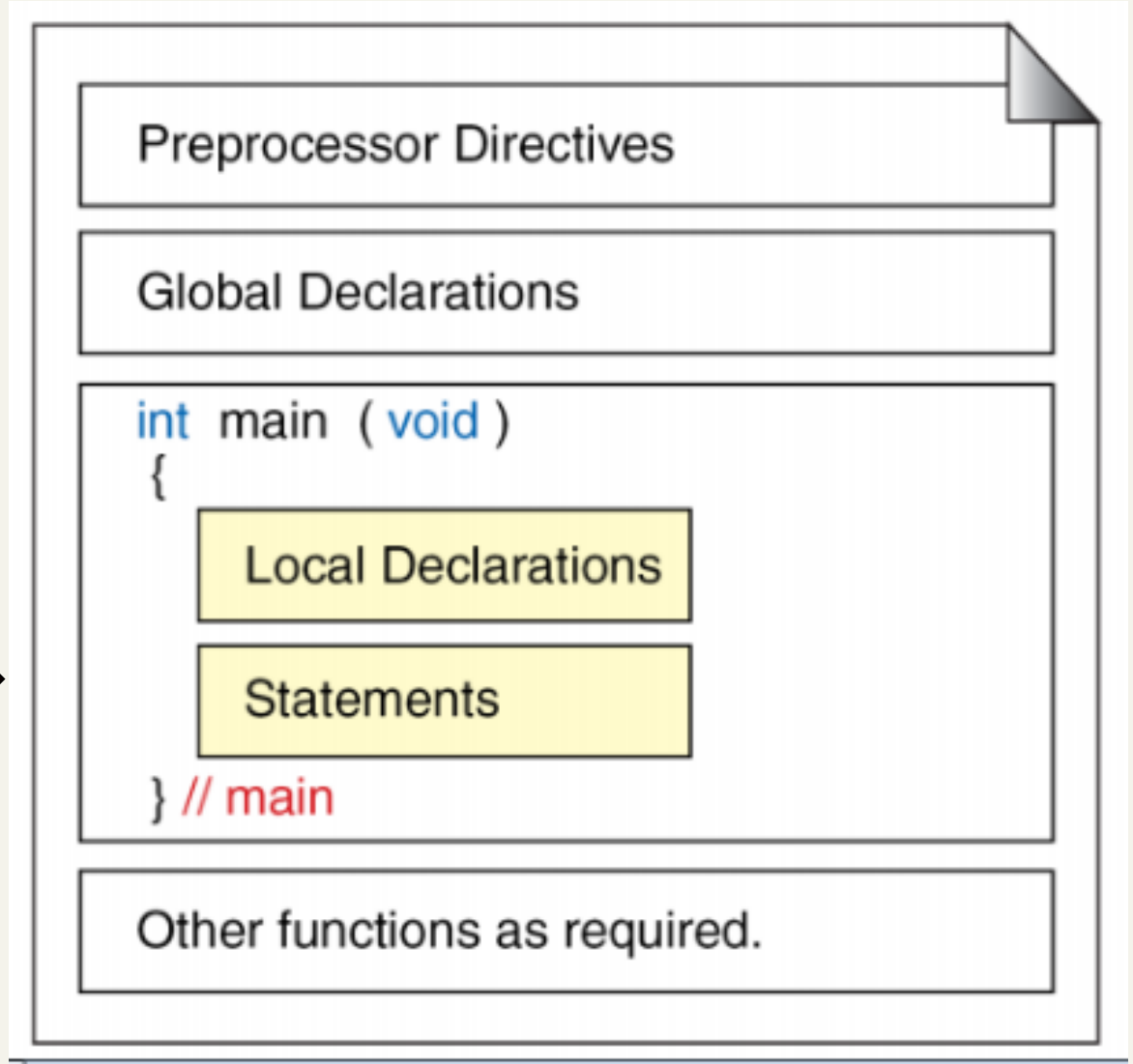
Cont…

- C is a **structured** programming language.
- It is considered a **high-level** language because it allows the programmer to concentrate on the problem at hand and not worry about the machine that the program will be using.
- While many languages claim to be **machine independent**, C is one of the closest to achieving that goal.
- That is another reason why it is used by software developers whose applications have to run on many **different hardware platforms**.

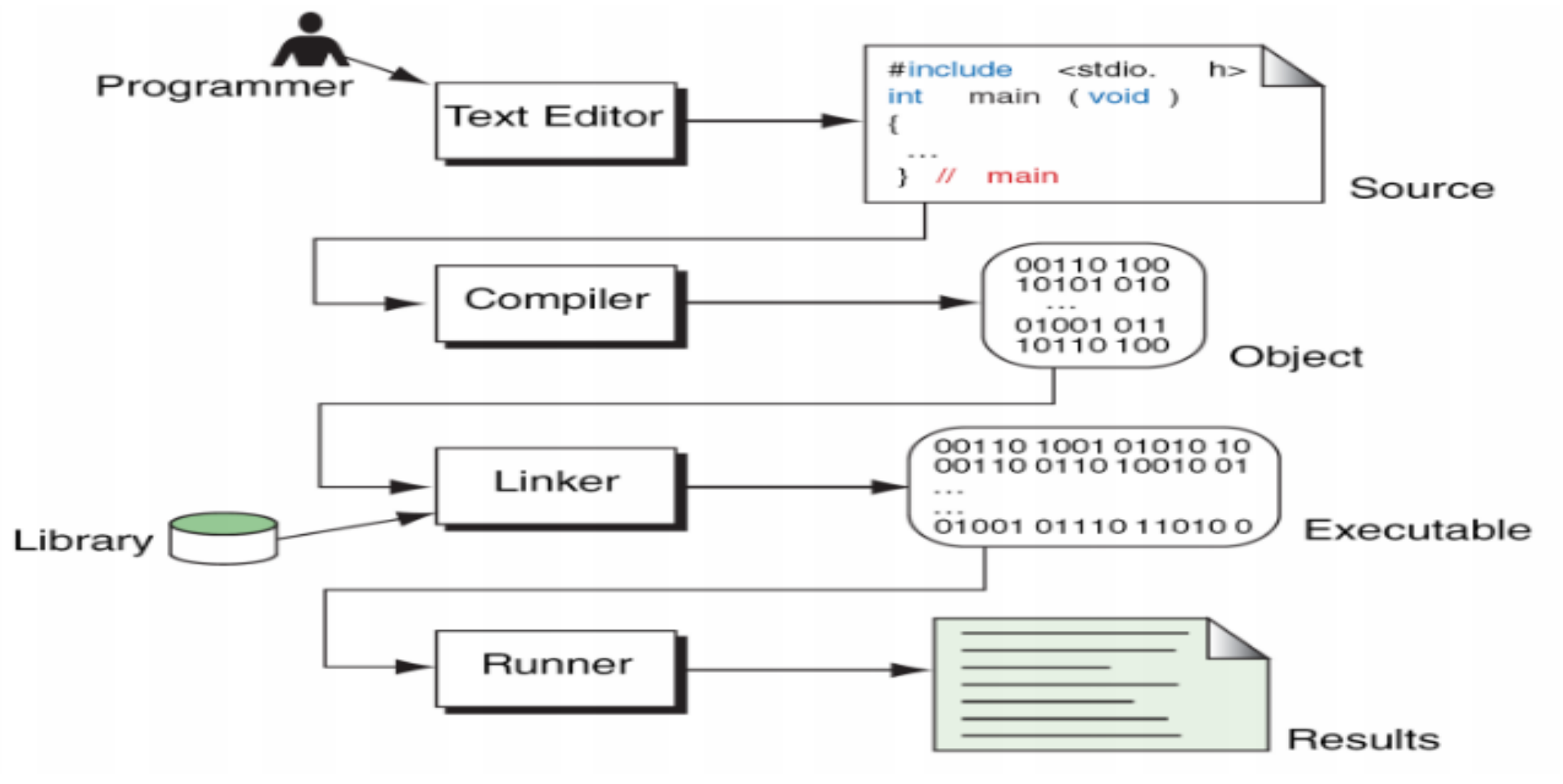
Structure of C Language

- It's time to write your first C program!

Structure of C Language



Creating and Running Programs



Greeting the World

```c
#include <stdio.h>

int main (void)
{
  printf("Hello World!\n");
  return 0;
} // main
```

Preprocessor directive to include standard input/output functions in the program.

Preprocessor Directive

- **# include**:
  - Is a preprocessor directive which directs to include the designated file which contains the declaration of library functions (pre defined).

- **stdio.h (standard input output)**:
  - A header file which contains declaration for standard input and output functions of the program. Like printf(), scanf()

Cont…

- When to use preprocessor directive
  - When one or more library functions are used, the corresponding header file where the information about these functions will be present are to be included.
  - When you want to make use of the functions (user-defined) present in a different program in your new program the source program of previous function has to be included.

Defining **main( )** function

- When a C program is executed, system first calls the **main( )** function,
  - thus a C program **must always contain** the function main() somewhere.

- A **function** definition has:

  heading

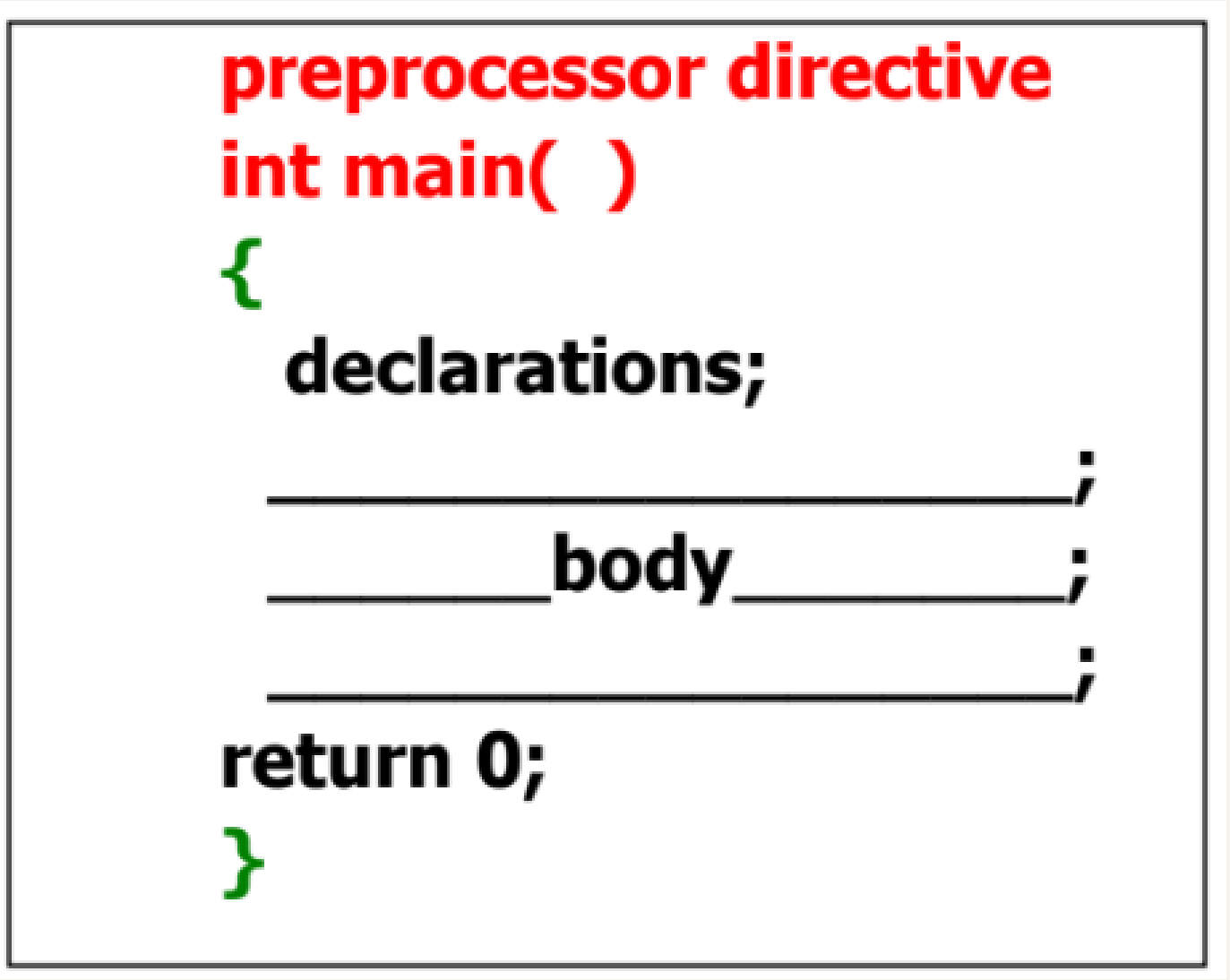
  {

  declarations ;

  statements;

  }

Basic Structure of a C program



Concept of Comment

- Comments are inserted in program to maintain **the clarity and for future** references. They help in easy **debugging**.

- Comments are **NOT compiled**, they are just for the **programmer** to maintain the **readability** of the source code.

- Comments are included as:

1. Single line comment: ***//*** ……..
2. Multiple line comment ***/**** .... */

Multiple line or paragraph comments

```
/* This is a block comment that
   covers two lines.                    */


/*
** It is a very common style to put the opening token
** on a line by itself, followed by the documentation
** and then the closing token on a separate line. Some
** programmers also like to put asterisks at the beginning
** of each line to clearly mark the comment.
*/
```

Line Comment examples

```
// This is a whole line comment

a = 5;                  // This is a partial line comment
```

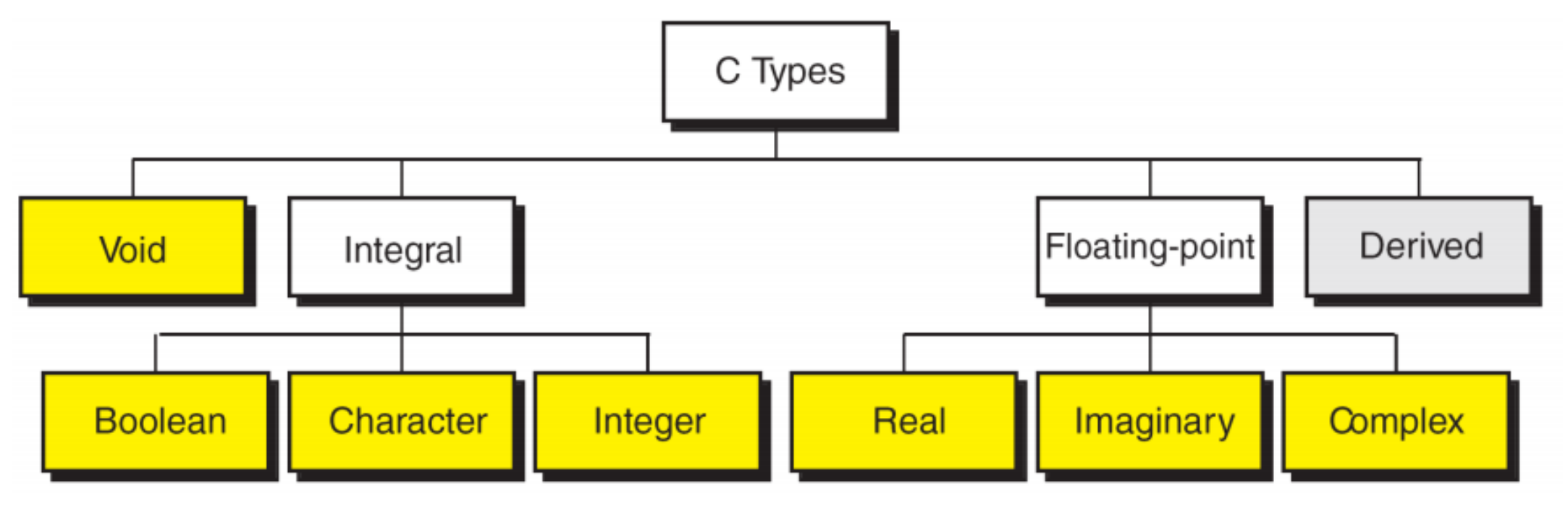
The Greeting Program

```
1  /* The greeting program. This program demonstrates
2     some of the components of a simple C program.
3        Written by:  your name here
4        Date:        date program written
5  */
6  #include <stdio.h>
7
8  int main (void)
9  {
10 // Local Declarations
11
12 // Statements
13
14    printf("Hello World!\n");
15
16    return 0;
17 } // main
```

Data Types

- Every program specifies a **set of operations** to be done **on** some **data** in a particular **sequence**.
- However, the **data** can be of many types such as a **number, real, character, string** etc.
- C supports many **data types** out of which the basic types are:
  - **int,**
  - **float ,**
  - **double and,**
  - **char**

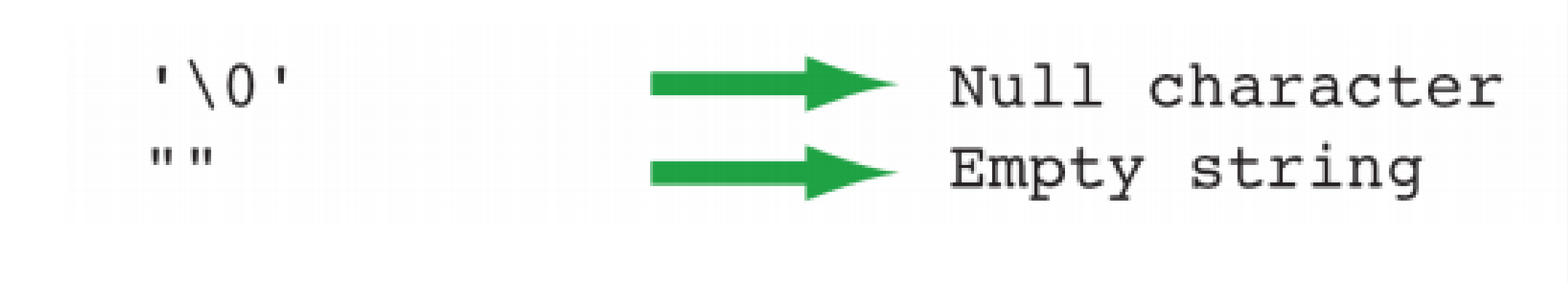
Data Types in C



Strings

```
""                                                    // A null string
"h"
"Hello World\n"
"HOW ARE YOU"
"Good Morning!"
L"This string contains wide characters."
```

Null Characters and Null Strings



Note

Use single quotes for character constants.
Use double quotes for string constants.

Notion of keywords

- Keywords are certain **reserved words**, that have standard **predefined meanings** in C.
- All keywords are in **lowercase**.

- Some keywords in C:

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

Identifiers

- One feature present in all computer languages is the **identifier**.
- Identifiers allow us to name **data** and other **objects** in the program.
- Each **identified** object in the computer is **stored** at a **unique address**.
- If we **didn’t** have identifiers that we could use to **symbolically** represent **data locations**, we would have to **know** and use object’s **addresses**.
- Instead, we simply **give** data identifiers and let the **compiler** keep track of where they are **physically located**.

Identifier - Variable

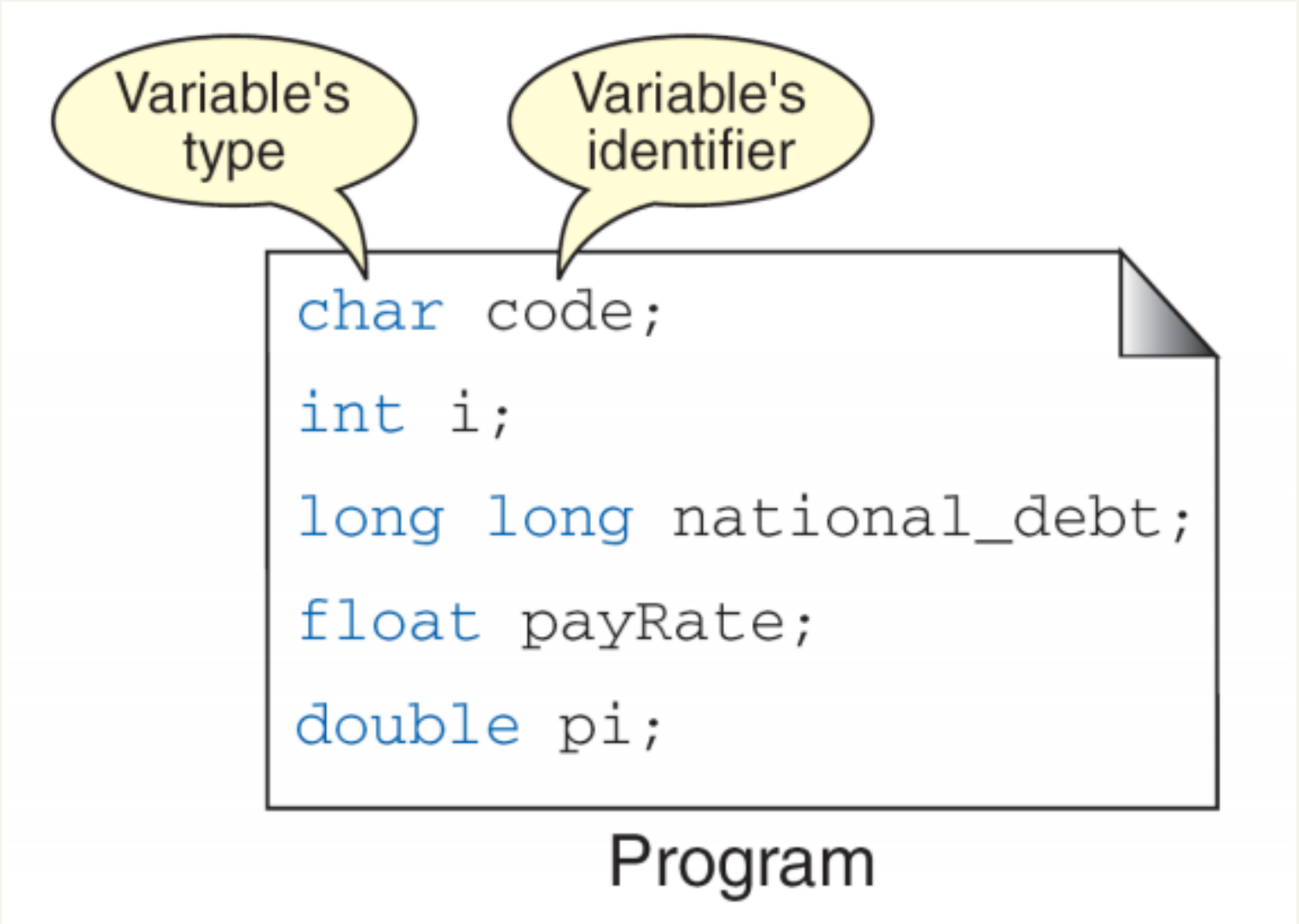
- In programming, a variable is a **container (storage area)** to hold data.
- To **indicate** the storage area, each variable should be given a **unique name (identifier).**
- Variable names are just the **symbolic** representation of a **memory location**.
- For example:
  - int playerScore = 95;
  - Here, **playerScore** is a variable of int type. Here, the variable is assigned an integer value 95.

Identifier - Variable

- The value of a variable can be changed, hence the name variable.

```
char ch = 'a';
// some code
ch = 'l';
```

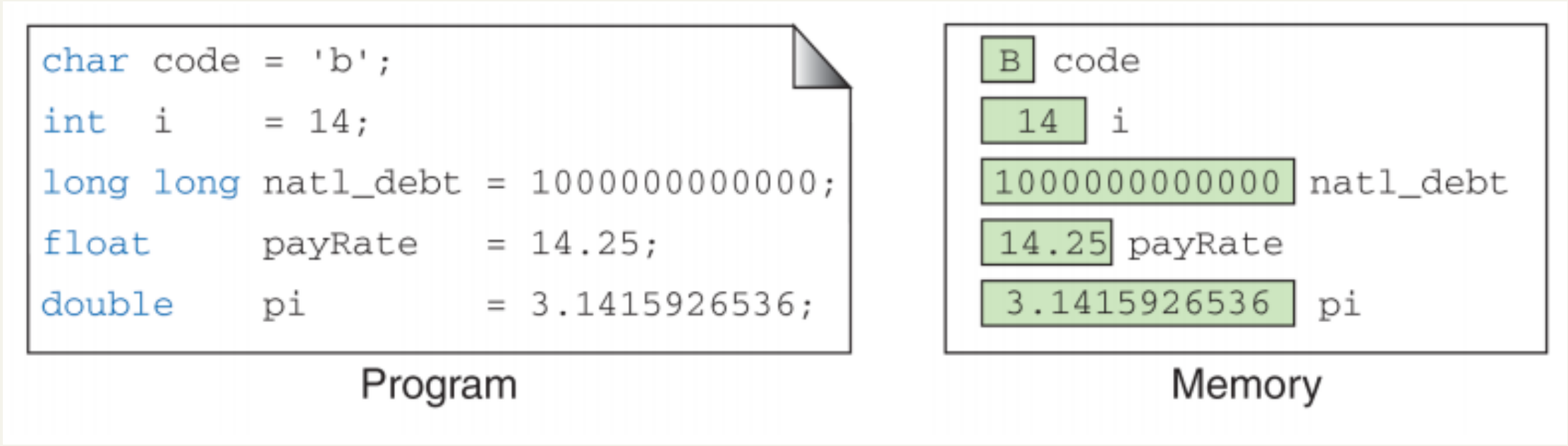
Identifier - Variable



Examples of Variable Declarations and Definitions

```
bool    fact;
short   maxItems;
long    long national_debt;
float   payRate;
double  tax;
float   complex voltage;
char    code, kind;
int     a, b;
```

Variable Initialization



Variables

Note

When a variable is defined, it is not initialized.
We must initialize any variable requiring
prescribed data when the function starts.

Identifier - Constants

- If you want to define a variable whose **value cannot** be changed during execution of the program, you can use the **const** keyword. This will create a constant.
  - For example,
    - **const** double PI = 3.14;
  - Notice, we have added keyword const.
  - Here, **PI** is a symbolic constant; its value **cannot** be changed.
- Now, if we do:-

  const double PI = 3.14;

  PI = 2.9; //Error
- You can also define a constant using the **#define** preprocessor directive.

Constants

- Character Constants

Note

A character constant is enclosed in single quotes.

Symbolic Names for Control Characters

ASCII Character	Symbolic Name
null character	`'\0'`
alert (bell)	`'\a'`
backspace	`'\b'`
horizontal tab	`'\t'`
newline	`'\n'`
vertical tab	`'\v'`
form feed	`'\f'`
carriage return	`'\r'`
single quote	`'\''`
double quote	`'\"'`
backslash	`'\\'`

Rules for Identifiers

- Uppercase letters are different than lowercase, example amount, Amount and AMOUNT all three are different identifiers.

1. First character must be alphabetic character or underscore.
2. Must consist only of alphabetic characters, digits, or underscores.
3. Maximum length can be 31 characters.
4. Cannot duplicate a keyword.

Cont…

Note

An identifier must start with a letter or underscore:
it may not have a space or a hyphen.

Cont…

Note

C is a case-sensitive language.

Examples of Identifiers

- Correct
- Correct
- Correct
- Correct
- Correct

```
a                   // Valid but poor style
student_name
_aSystemName
_Bool               // Boolean System id
INT_MIN             // System Defined Value
```

Examples of incorrect Identifiers

```
$sum            // $ is illegal

2names          // First char digit

sum-salary      // Contains hyphen

stdnt Nmbr      // Contains spaces

int             // Keyword
```

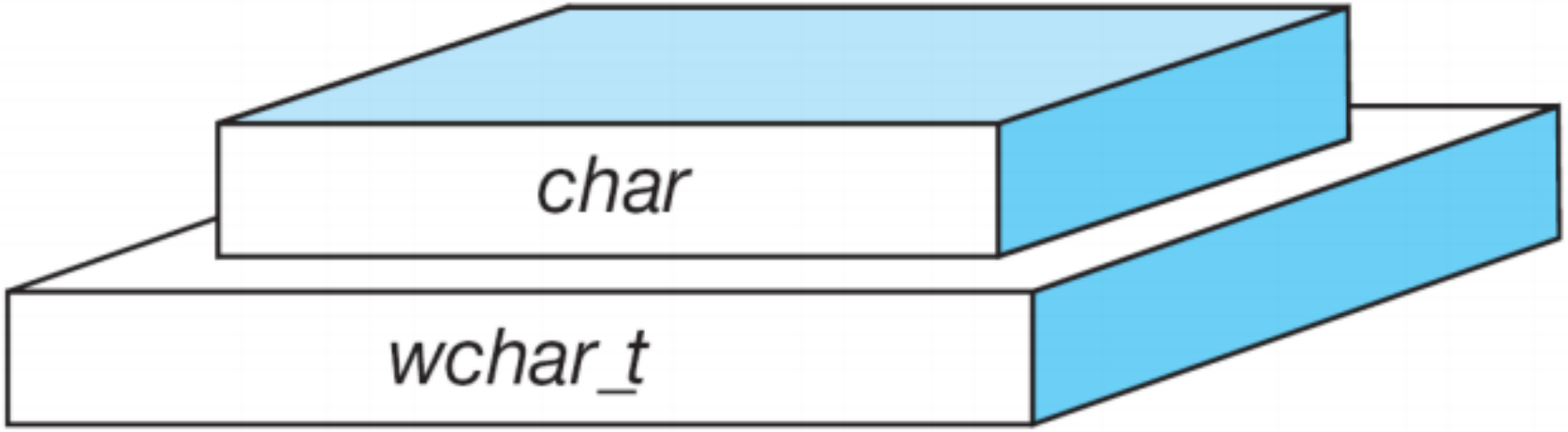
Format specifiers

- There are several format specifiers –
  - The one you use should depend on the **type of the variable** you wish to **print out**.
  - The common ones are as follows:

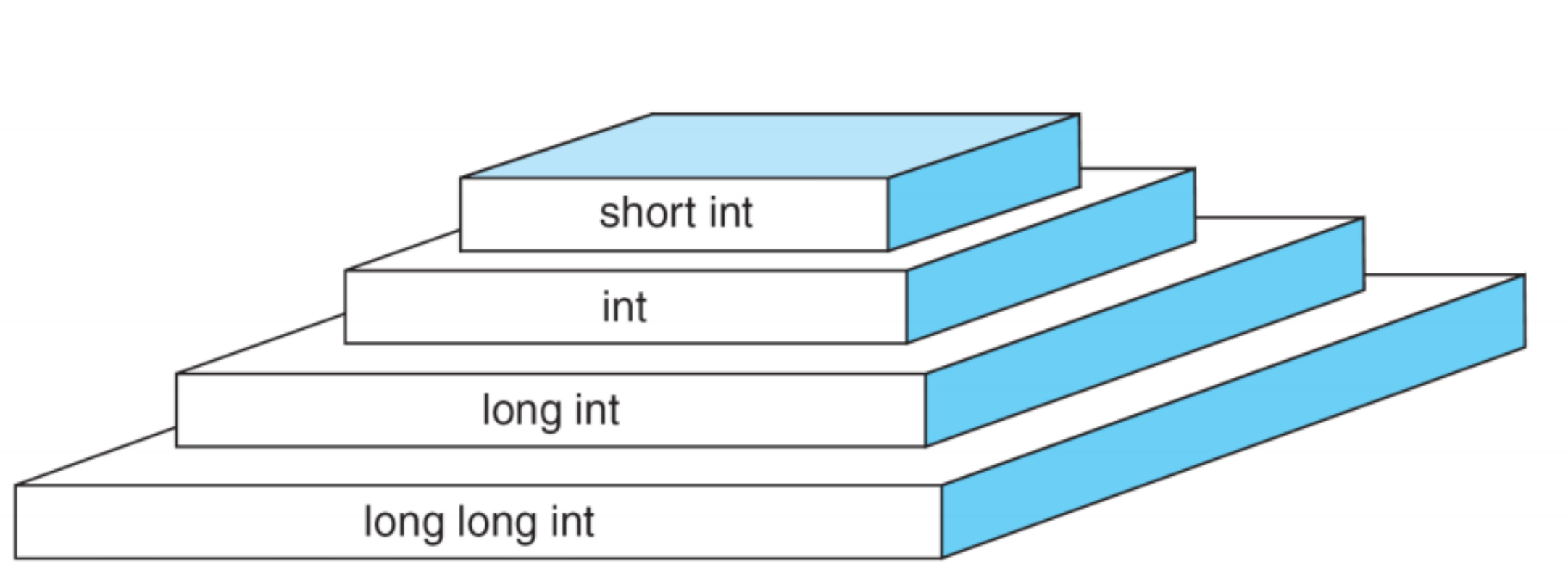
Format Specifier	Type
%d	int
%c	char
%f	float
%lf	double
%s	string

To display a number in scientific notation,use %e.
To display a percentage sign,use %%

Character Types



Integral Types



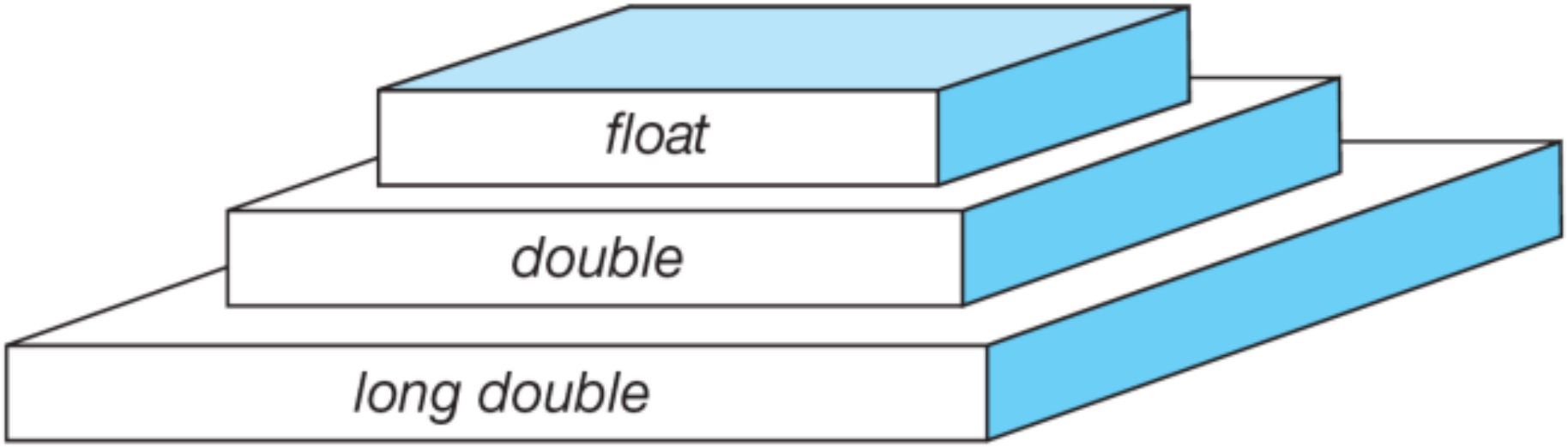
Note

sizeof (short) ≤ sizeof (int) ≤ sizeof (long) ≤ sizeof (long long)

Typical Integer Sizes and Values for Signed Integers

Type	Byte Size	Minimum Value	Maximum Value
short int	2	–32,768	32,767
int	4	–2,147,483,648	2,147,483,647
long int	4	–2,147,483,648	2,147,483,647
long long int	8	–9,223,372,036,854,775,807	9,223,372,036,854,775,806

Floating point Types



Note

sizeof (float) ≤ sizeof (double) ≤ sizeof (long double)

Range of real constants expressed in exponential form is -3.4e38 to 3.4e38.

Type Summary

Category	Type	C Implementation
Void	Void	*void*
Integral	Boolean	*bool*
	Character	*char, wchar_t*
	Integer	*short int, int, long int, long long int*
Floating-Point	Real	*float, double, long double*
	Imaginary	*float imaginary, double imaginary, long double imaginary*
	Complex	*float complex, double complex, long double complex*

Expression and Operators

- The symbols which are used to perform **logical and mathematical operations** in a C program are called C **operators**.
- These C operators **join individual constants and variables** to form **expressions**.
- Operators, functions, constants and variables are combined together to form **expressions**.
- Consider the expression **A + B * 5**. where, +, * are **operators**, A, B are **variables**, 5 is **constant** and A + B * 5 is an **expression**.

Types of C operators

- Arithmetic operators
  - Addition (+), subtraction(-), multiplication(*), division(/) and modulus(%)
- Assignment operators
  - Simple assignment operator (=) { Ex: a = 3)
  - Compound assignment operators (+=, -=, *=, /=, %=, &=, ^=)
- Relational operators
- Logical operators (logical AND (&&), logical OR (||) and logical NOT (!))
- Bit wise operators (&-bitwise AND, |-bitwise OR, ~-bitwise NOT, ^-XOR, <<-left shift and >>-right shift)
- Conditional operators (ternary operators) (Condition? true_value: false_value)
- Increment/decrement operators (++, --)
- Special operators (&, *, sizeof( ))

Compound operators Examples

Operators	Example/Description
=	sum = 10; 10 is assigned to variable sum
+=	sum += 10; This is same as sum = sum + 10
-=	sum -= 10; This is same as sum = sum – 10
*=	sum *= 10; This is same as sum = sum * 10
/=	sum /= 10; This is same as sum = sum / 10
%=	sum %= 10; This is same as sum = sum % 10
&=	sum&=10; This is same as sum = sum & 10
^=	sum ^= 10; This is same as sum = sum ^ 10

Relational Operators Examples

Operators	Example/Description
>	x > y (x is greater than y)
<	x < y (x is less than y)
>=	x >= y (x is greater than or equal to y)
<=	x <= y (x is less than or equal to y)
==	x == y (x is equal to y)
!=	x != y (x is not equal to y)

Logical Operators Examples

Operators	Example/Description
&& (logical AND)	(x>5)&&(y<5) It returns true when both conditions are true
|| (logical OR)	(x>=10)||(y>=10) It returns true when at-least one of the condition is true
! (logical NOT)	!((x>5)&&(y<5)) It reverses the state of the operand "((x>5) && (y<5))" If "((x>5) && (y<5))" is true, logical NOT operator makes it false

Conditional Operators Example

- Ex:
  - (A > 100 ? 0 : 1);

- In above example, if A is greater than 100, 0 is returned else 1 is returned. This is equal to if else conditional statements.

Increment / decrement Operators

- Example:
  Increment operator : ++ i ; i ++ ;
  Decrement operator : – – i ; i – – ;
- ++i => i=i+1 [Note: it is pre increment, means the value of i is incremented here only]
- i++ => i=i+1 [Note: it is post increment, means the value of I is incremented in the next statement]

Increment / decrement Operators Examples

Operator	Operator/Description
Pre increment operator (++i)	value of i is incremented before assigning it to the variable i
Post increment operator (i++)	value of i is incremented after assigning it to the variable i
Pre decrement operator (–i)	value of i is decremented before assigning it to the variable i
Post decrement operator (i–)	value of i is decremented after assigning it to variable i

Special Operators Examples

Operators	Description
&	This is used to get the address of the variable. Example : &a will give address of a.
*	This is used as pointer to a variable. Example : * a where, * is pointer to the variable a.
Sizeof ()	This gives the size of the variable. Example : size of (char) will give us 1.