C - Fundamentals

Pratik Shah
II – Sem CSE (DIV A)
02/01/25 to 21/04/25

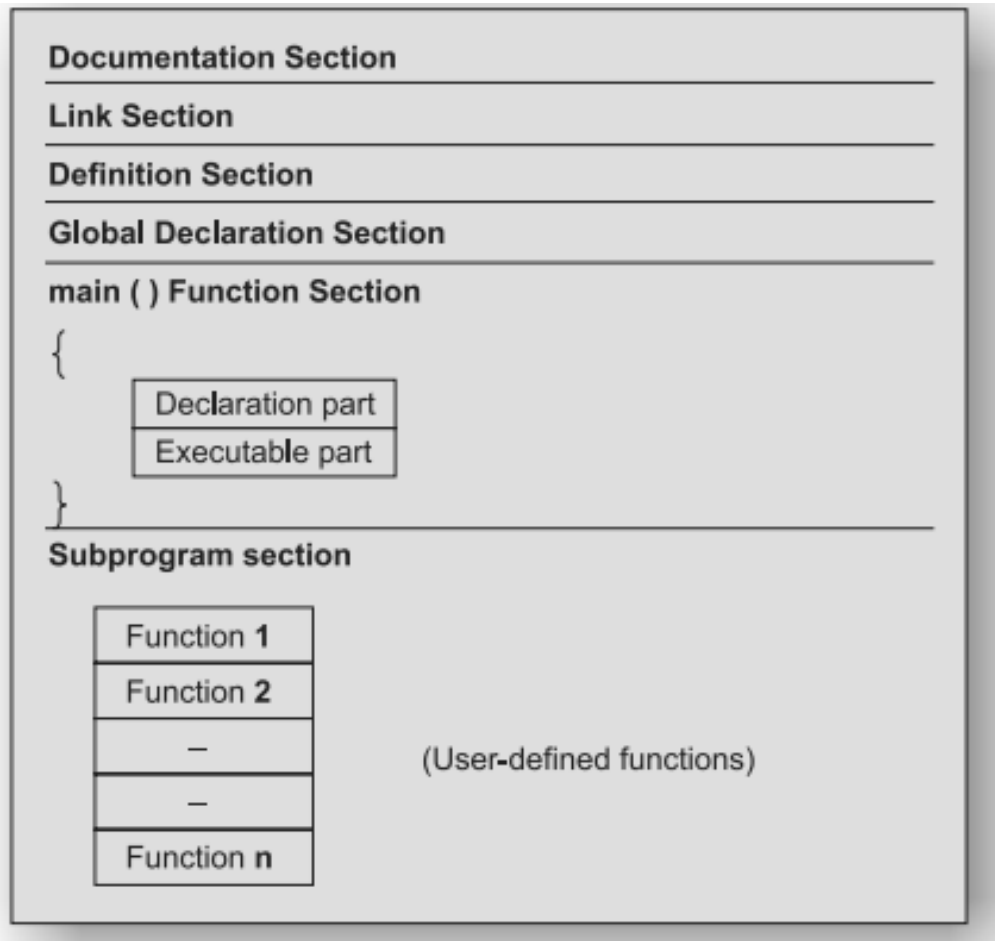
Student will be able to :

	Introduction to Programming and C fundamentals 1.1 Algorithms, Flowchart, 1.2 Programming Languages, Types of Languages
I	1.3 Basic Structure of C programming 1.4 Process of Executing C program 1.5 Character Sets, Keywords 1.6 Data types: int, char, float 1.7 Library I/O Functions 1.8 Identifiers, Constants, Declaration, Storage classes 1.9 Data input and output formatting

Importance of C

- It is a **robust language** whose rich set of built-in functions and operators can be used to write any complex program.
- Programs written in C are **efficient and fast**.
- There are only **32 keywords** in ANSI C and its strength lies in its built-in functions.
- C is highly **portable.**(This means that C programs written for one computer can be run on another with little or no modification. Portability is important if we plan to use a new computer with a different operating system)
- C language is well suited for **structured programming**
- Another important feature of C is its ability to **extend itself**. A C program is basically a collection of functions that are supported by the C library. We can continuously add our own functions to C library.
- With the availability of a large number of functions, the programming task becomes **simple.**

Basic structure of C programs



The **subprogram section** contains all the user-defined functions that are called in the main function. User-defined functions are generally placed immediately after the main function, although they may appear in any order.

The **documentation section** consists of a set of comment lines giving the name of the program, the author and other details, which the programmer would like to use later.

The **link section** provides instructions to the compiler to link functions from the system library.

The **definition section** defines all symbolic constants.

There are some variables that are used in more than one function. Such variables are called **global variables** and are declared in the global declaration section that is outside of all the functions.

Every C program must have one **main() function section**. This section contains two parts, declaration part and executable part.
The declaration part declares all the variables used in the executable part. There is at least one statement in the executable part. These two parts must appear between the opening and the closing braces. The program execution begins at the opening brace and ends at the closing brace. The closing brace of the main function section is the logical end of the program. All statements in the declaration and executable parts end with a semicolon(;).

Executing a ‘C’ program

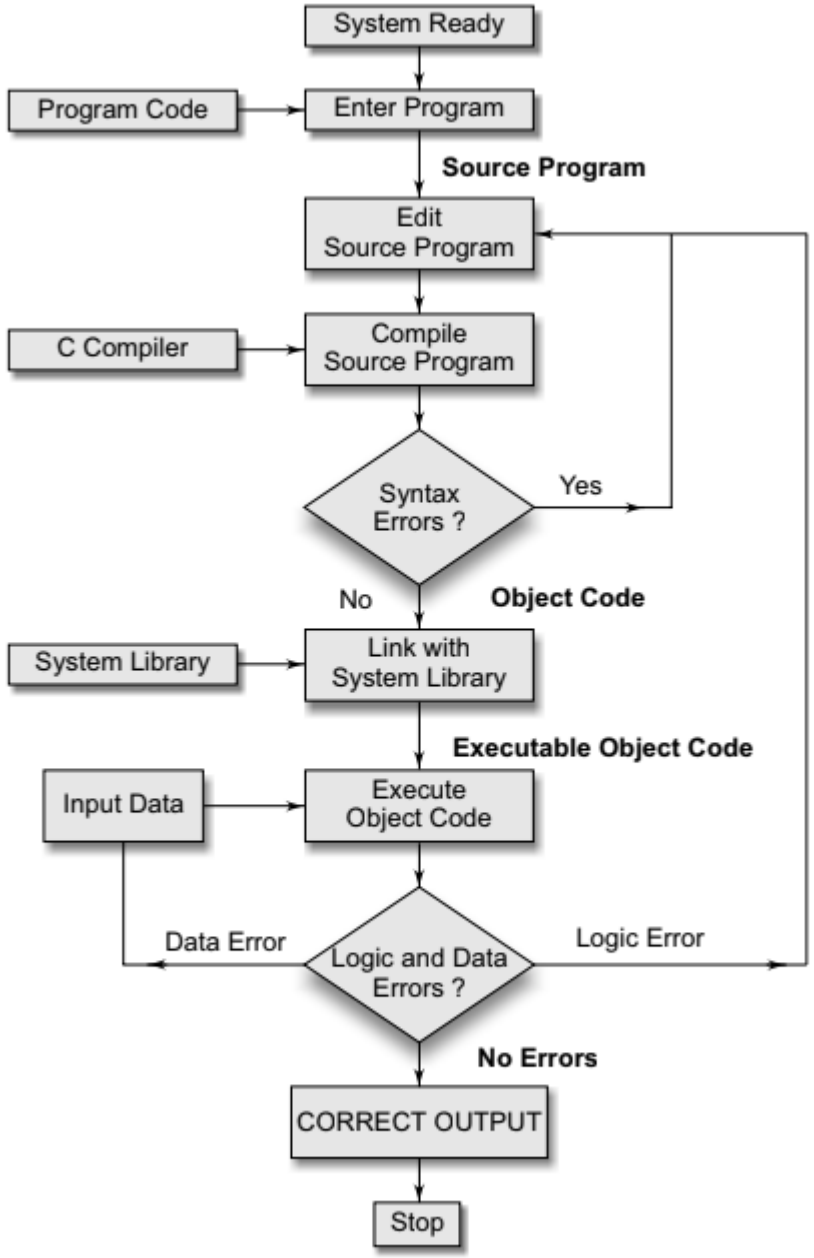
Executing a program written in C involves a series of steps.

1. Creating the program

2. Compiling the program

3. Linking the program with functions that are needed from the C library

4. Executing the program



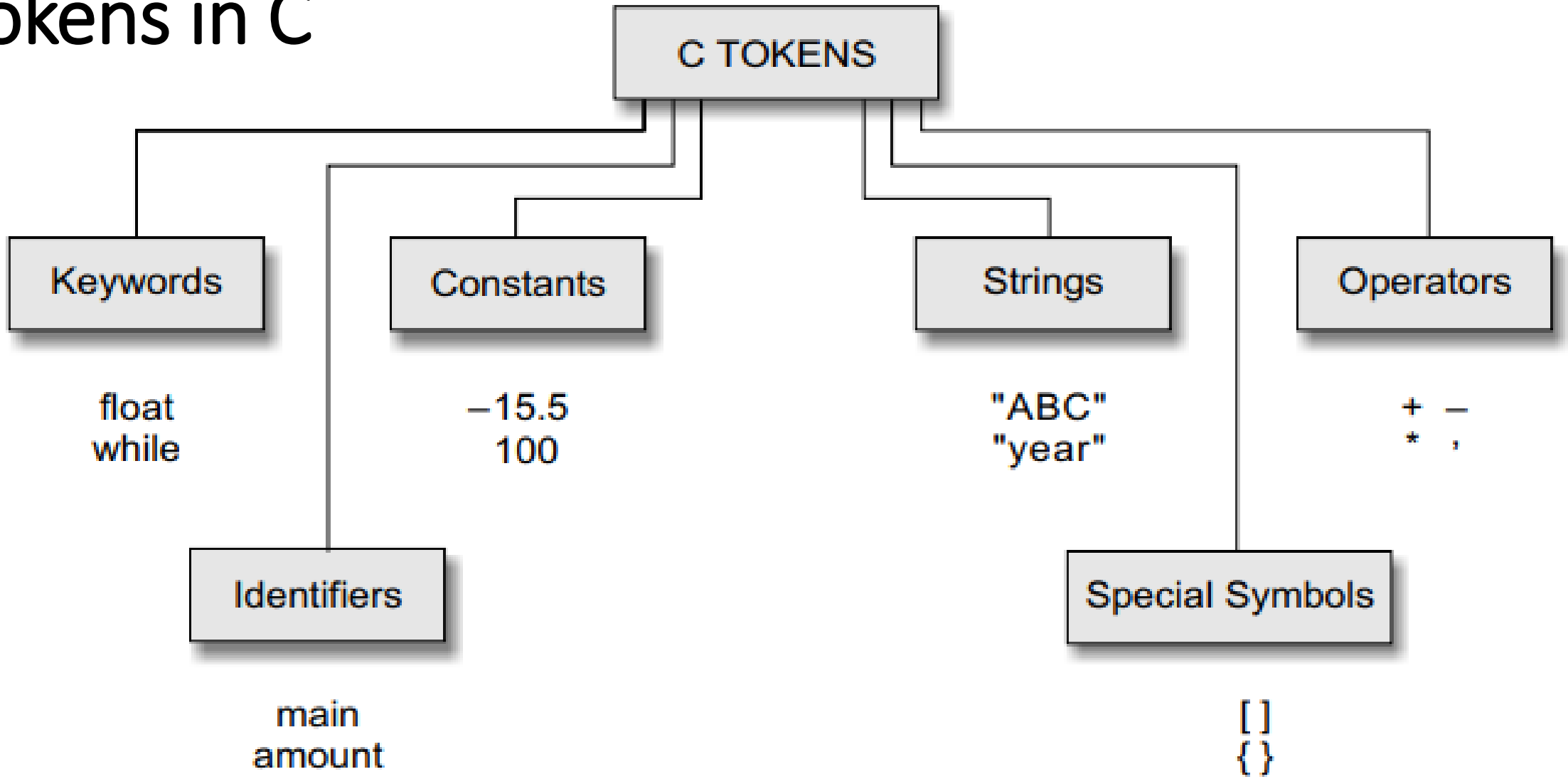
Process of compiling and runnig a C program

Character Set

- The characters that can be used to form words, numbers and expressions depend upon the computer on which the program is run.
- The characters in C are grouped into the following categories:
  - Letters
  - Digits
  - Special characters
  - White spaces

Letters		*Digits*
Uppercase A.....Z		All decimal digits 0 9
Lowercase a.....z		
	Special Characters	
, comma		& ampersand
. period		^ caret
; semicolon		* asterisk
: colon		– minus sign
? question mark		+ plus sign
‘ apostrophe		< opening angle bracket
“ quotation mark		(or less than sign)
! exclamation mark		> closing angle bracket
| vertical bar		(or greater than sign)
/ slash		(left parenthesis
\ backslash		) right parenthesis
~ tilde		[left bracket
_ under score		] right bracket
$ dollar sign		{ left brace
% percent sign		} right brace
		# number sign
	White Spaces	
	Blank space	
	Horizontal tab	
	Carriage return	
	New line	
	Form feed	

Tokens in C



C tokens and examples

- In a passage of text, individual words and punctuation marks are called **tokens.**
- A C program consists of various tokens and a token is either a keyword, an identifier, a constant, a string literal, or a symbol. For example, the following C statement consists of five tokens −
  - printf("Hello, World! \n");

The individual tokens are −

printf

(

"Hello, World! \n"

)

;

Semicolons

- In a C program, the semicolon is a statement terminator. That is, each individual statement must be ended with a semicolon. It indicates the end of one logical entity.
- Given below are two different statements −
  - printf("Hello, World! \n");
  - c=a+b;

Comments

- Comments are like helping text in your C program and they are ignored by the compiler. They start with /* and terminate with the characters */ as shown below –

```
/* my first program in C */
```

- Single line comment : they start with //
  - //my first program

Keywords

- The following list shows the reserved words in C. These reserved words may not be used as constants or variables or any other identifier names.

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

Identifiers

- A C identifier is a name used to identify a variable, function, or any other user-defined item. An identifier starts with a letter A to Z, a to z, or an underscore '_' followed by zero or more letters, underscores, and digits (0 to 9).
- C does not allow punctuation characters such as @, $, and % within identifiers. C is a **case-sensitive** programming language.
  Thus, *Manpower* and *manpower* are two different identifiers in C.
  Here are some examples of acceptable identifiers –

  mohd zara abc move_name a_123

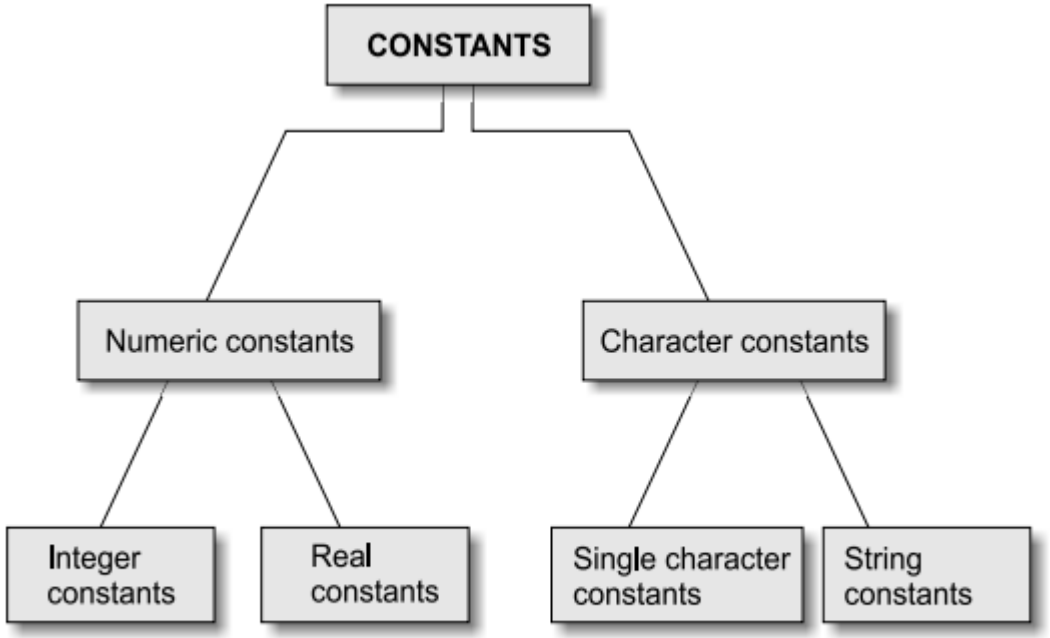
  myname50 _temp j a23b9 retVal

Rules for Identifiers

- First character must be an alphabet (or underscore).
- Must consist of only letters, digits or underscore.
- Only first 31 characters are significant.
- Cannot use a keyword.
- Must not contain white space

Constants

- Constants in C refer to fixed values that do not change during the execution of a program.
- C supports several types of constants as illustrated in diagram



Basic types of C constants

Backslash character constants

- C supports some special backslash character constants that are used in output functions.

Backslash Character Constants

Constant	*Meaning*
'\a'	audible alert (bell)
'\b'	back space
'\f'	form feed
'\n'	new line
'\r'	carriage return
'\t'	horizontal tab
'\v'	vertical tab
'\''	single quote
'\"'	double quote
'\?'	question mark
'\\'	backslash
'10'	null

C - Variables

- A variable is a data name that may be used to store a data value.
- A variable name can be chosen by the programmer in a meaningful way so as to reflect its function or nature in the program.
- Some examples of such names are:
  - Average
  - height
  - Total
  - Counter_1
  - class_strength

- variable names may consist of **letters, digits, and the underscore(_) character**, subject to the following conditions:
  - They must **begin** with a **letter**.
  - Some systems permit underscore as the first character.
  - ANSI standard recognizes a length of 31 characters. However, **length** should **not** be normally more than **eight characters**, since only the first eight characters are treated as significant by many compilers. (In C99, at least 63 characters are significant.)
  - Uppercase and lowercase are **significant.** That is, the variable Total is not the same as total or TOTAL.
  - It should **not** be a **keyword**.
  - **White space** is **not** allowed.

Some valid declarations are shown here

- int i, j, k;
- char c, ch;
- float f, salary;
- double d;

- The line **int i, j, k;** declares and defines the variables i, j, and k; which instruct the compiler to create variables named i, j and k of type int.

- Variables can be initialized (assigned an initial value) in their declaration. The initializer consists of an equal sign followed by a constant expression as follows −
  - type variable_name = value;
  - Some examples are −
  - int d = 3, f = 5; // definition and initializing d and f.
  - char x = 'x'; // the variable x has the value 'x'.

Whitespace in C

- A line containing only whitespace, possibly with a comment, is known as a blank line, and a C compiler totally ignores it.
- Whitespace is the term used in C to describe blanks, tabs, newline characters and comments.
- Whitespace separates one part of a statement from another and enables the compiler to identify where one element in a statement, such as int, ends and the next element begins.

- Therefore, in the following statement −
  - **int age;**

  there must be at least one whitespace character (usually a space) between int and age for the compiler to be able to distinguish them.

  On the other hand, in the following statement −
  - **fruit = apples + oranges;** // get the total fruit

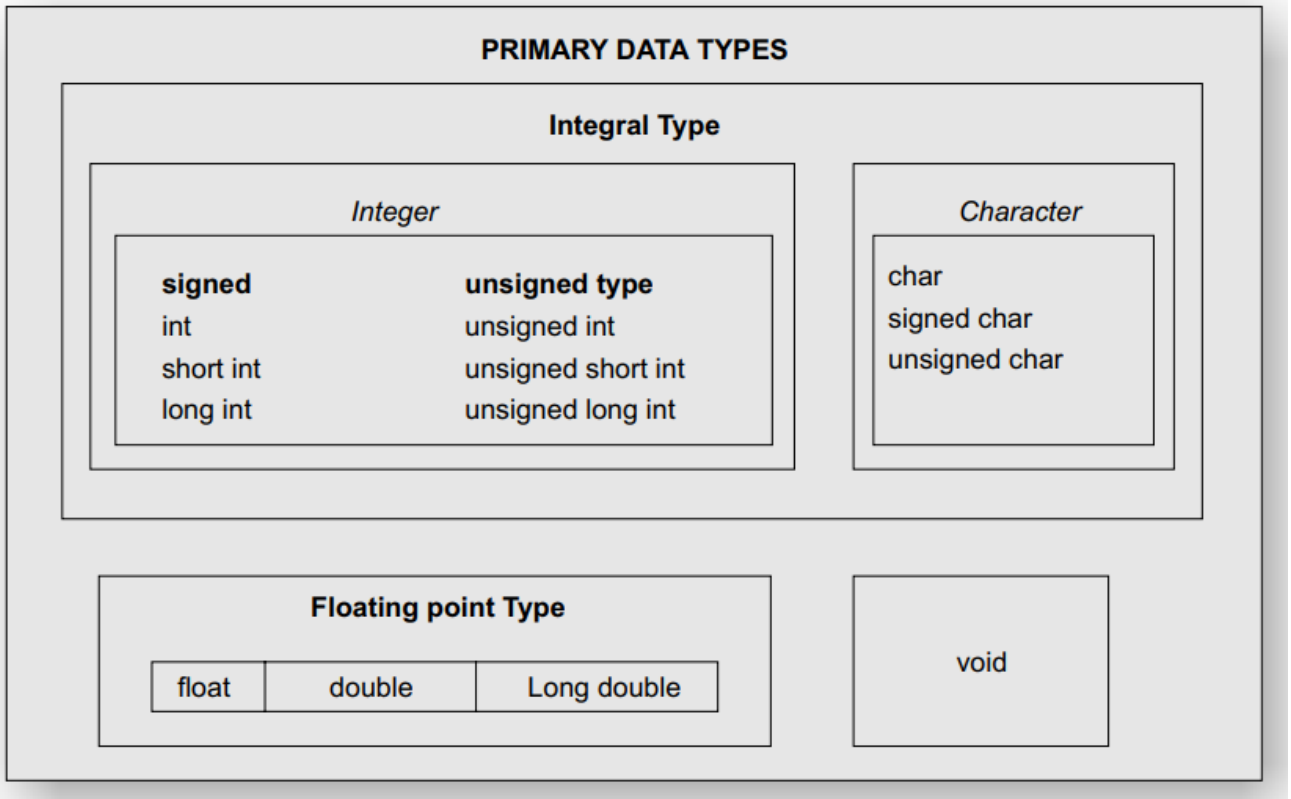
  no whitespace characters are necessary between fruit and =, or between = and apples, although you are free to include some if you wish to increase readability.

C - Data Types

- Data types in c refer to an extensive system used for declaring variables or functions of different types.
- The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.
- ANSI C supports **three** classes of **data types**:
  - **Primary (or fundamental) data types**
  - **Derived data types**
  - **User-defined data types**

Primary Datatypes

- All C compilers support **five** fundamental data types, namely integer (**int**), character (**char**), floating point (**float**), double-precision floating point (**double**) and **void**.



Primary data types in C

Size and Range of Data Types on a 16-bit Machine

Type	*Size (bits)*	*Range*
char or signed char	8	–128 to 127
unsigned char	8	0 to 255
int or signed int	16	–32,768 to 32,767
unsigned int	16	0 to 65535
short int or signed short int	8	–128 to 127
unsigned short int	8	0 to 255
long int or signed long int	32	–2,147,483,648 to 2,147,483,647
unsigned long int	32	0 to 4,294,967,295
float	32	3.4E – 38 to 3.4E + 38
double	64	1.7E – 308 to 1.7E + 308
long double	80	3.4E – 4932 to 1.1E + 4932

Derived Datatypes

- **Derived Datatypes** are composed of fundamental datatypes;
- They are derived from the fundamental data types.
- Therefore, they have some additional characteristics and properties other than that of fundamental data types.
- Programmers can modify or redefine the derived datatypes.
- Some common examples of derived datatypes include **Arrays, Functions, Pointers,** etc.

User-defined datatypes

- User-defined data types are created by the user using a combination of fundamental and derived data types in the C programming language.
- Some common examples of User-defined datatypes include **Structure, Union, Typedef, enum**

Declaration of Storage Class

- Variables in C can have not only data type but also storage class that provides information about their location and visibility.
- The storage class decides the portion of the program within which the variables are recognized.

- Consider the following example:

```
/* Example of storage classes */
int   m;
main()
{
   int i;
   float balance;
   ....
   ....
  function1();
}
function1()
{
   int i;
   float sum;
   ....
   ....
   }
```

The **variable m** which has been declared before the main is called **global variable**

It can be used in all the functions in the program.

It need not be declared in other functions.

A global variable is also known as an external variable.

The variables **i, balance and sum** are called **local variables** because they are declared inside a function.

Local variables are visible and meaningful only inside the functions in which they are declared. They are not known to other functions.

Note that the **variable i** has been declared in both the functions. Any change in the value of i in one function does not affect its value in the other.

- C provides a variety of storage class specifiers that can be used to declare explicitly the scope and lifetime of variables.
- The concepts of scope and lifetime are important only in multifunction and multiple file programs and therefore the storage classes are considered in detail later when functions are discussed.
- For now, remember that there are four storage class specifiers

  (**auto, register, static, and extern**)

Storage Classes and Their Meaning

Storage class	*Meaning*
auto	Local variable known only to the function in which it is declared. *Default is auto.*
static	Local variable which exists and retains its value even after the control is transferred to the calling function.
extern	Global variable known to all functions in the file.
register	Local variable which is stored in the register.

Automatic

- Automatic variables are allocated memory automatically at runtime.
- The visibility of the automatic variables is limited to the block in which they are defined.
- The scope of the automatic variables is limited to the block in which they are defined. The automatic variables are initialized to garbage by default.
- The memory assigned to automatic variables gets freed upon exiting from the block.
- The keyword used for defining automatic variables is auto.
- Every local variable is automatic in C by default.

```c
#include <stdio.h>
int main()
{
int a; //auto
char b;
float c;
printf("%d %c %f",a,b,c); // printing initial default value of automatic variables a, b, and c.
return 0;
}
```

Output:

```
garbage garbage garbage
```

Static

- The variables defined as static specifier can hold their value between the multiple function calls.
- Static local variables are visible only to the function or the block in which they are defined.
- A same static variable can be declared many times but can be assigned at only one time.
- Default initial value of the static integral variable is 0 otherwise null.
- The visibility of the static global variable is limited to the file in which it has declared.
- The keyword used to define static variable is static.

Example 1

```
#include<stdio.h>
static char c;
static int i;
static float f;
static char s[100];
void main ()
{
printf("%d %d %f %s",c,i,f); // the initial default value of c, i, and f will be printed.
}
```

Output:

0 0 0.000000 (null)

Register

- The variables defined as the register is allocated the memory into the CPU registers depending upon the size of the memory remaining in the CPU.
- We can not dereference the register variables, i.e., we can not use &operator for the register variable.
- The access time of the register variables is faster than the automatic variables.
- The initial default value of the register local variables is 0.
- The register keyword is used for the variable which should be stored in the CPU register. However, it is compilers choice whether or not; the variables can be stored in the register.
- We can store pointers into the register, i.e., a register can store the address of a variable.
- Static variables can not be stored into the register since we can not use more than one storage specifier for the same variable.

Example 1

```
#include <stdio.h>
int main()
{
register int a; // variable a is allocated memory in the CPU register. The initial default value of a is 0.
printf("%d",a);
}
```

Output:

```
0
```

External

- The external storage class is used to tell the compiler that the variable defined as extern is declared with an external linkage elsewhere in the program.
- The variables declared as extern are not allocated any memory. It is only declaration and intended to specify that the variable is declared elsewhere in the program.
- The default initial value of external integral type is 0 otherwise null.
- We can only initialize the extern variable globally, i.e., we can not initialize the external variable within any block or method.
- An external variable can be declared many times but can be initialized at only once.
- If a variable is declared as external then the compiler searches for that variable to be initialized somewhere in the program which may be extern or static. If it is not, then the compiler will show an error.

Example 1

```c
#include <stdio.h>
int main()
{
extern int a;
printf("%d",a);
}
```

Output

```
main.c:(.text+0x6): undefined reference to `a'
collect2: error: ld returned 1 exit status
```

C library functions

- Library functions are built-in functions that are grouped together and placed in a common location called library.
- Each function here performs a specific operation. We can use this library functions to get the pre-defined output.
- All C standard library functions are declared by using many header files.
- These library functions are created at the time of designing the compilers.
- We include the header files in our C program by using **#include<filename.h>**. Whenever the program is run and executed, the related files are included in the C program.

Header File Functions

- Some of the header file functions are as follows –
- **stdio.h** – It is a standard i/o header file in which Input/output functions are declared
- **conio.h** – This is a console input/output header file.
- **string.h** – All string related functions are in this header file.
- **stdlib.h** – This file contains common functions which are used in the C programs.
- **math.h** – All functions related to mathematics are in this header file.
- **time.h** – This file contains time and clock related functions.Built functions in stdio.h

Built functions in stdio.h

1	**printf()** This function is used to print the all char, int, float, string etc., values onto the output screen.
2	**scanf()** This function is used to read data from keyboard.
3	**getc()** It reads character from file.
4	**gets()** It reads line from keyboard.
5	**getchar()** It reads character from keyboard.
6	**puts()** It writes line to o/p screen.
7	**putchar()** It writes a character to screen.
8	**fopen()** All file handling functions are defined in stdio.h header file.
9	**fclose()** Closes an opened file.
10	**getw()** Reads an integer from file.
11	**putw()** Writes an integer to file.
12	**fgetc()** Reads a character from file.
13	**putc()** Writes a character to file.
14	**fputc()** Writes a character to file.
15	**fgets()** Reads string from a file, one line at a time.
16	**f puts()** Writes string to a file.
17	**feof()** Finds end of file.

18	**fgetchar** Reads a character from keyboard.
19	**fgetc()** Reads a character from file.
20	**fprintf()** Writes formatted data to a file.
21	**fscanf()** Reads formatted data from a file.
22	**fputchar** Writes a character from keyboard.
23	**fseek()** Moves file pointer to given location.
24	**SEEK_SET** Moves file pointer at the beginning of the file.
25	**SEEK_CUR** Moves file pointer at given location.
26	**SEEK_END** Moves file pointer at the end of file.
27	**ftell()** Gives current position of file pointer.

28	**rewind()** Moves file pointer to the beginning of the file.
29	**putc()** Writes a character to file.
30	**sprint()** Writes formatted output to string.
31	**sscanf()** Reads formatted input from a string.
32	**remove()** Deletes a file.
33	**flush()** Flushes a file.

https://www.tutorialspoint.com/what-are-the-c-library-functions

Data Output Formatting

- To display any message or value on output screen, then we use **printf()**
- printf and PRINTF are not the same.
- In C, everything is written in lowercase letters.
- **Syntax :** **printf(“Your message here”);**
- **Example :** **printf(“Hello World”);**

- **To print the value of variable :** **printf(“Control string”, name of variable);**
- **Example :** **printf(“%d”,a);**

Data Input Formatting

- Another way of giving values to variables is to input data through keyboard using the **scanf function.**
- The general format of scanf is as follows:

```
scanf("control string", &variable1,&variable2,....);
```

- The control string contains the format of data being received.
- The ampersand symbol & before each variable name is an operator that specifies the variable name's address.
- We must always use this operator, otherwise unexpected results may occur.

Example

scanf("%d", &number);

- When this statement is encountered by the computer, the execution stops and waits for the value of the variable number to be typed in. Since the control string "%d" specifies that an integer value is to be read from the terminal, we have to type in the value in integer form.
- Once the number is typed in and the 'Enter' Key is pressed, the computer then proceeds to the next statement.
- Thus, the use of scanf provides an interactive feature and makes the program 'user friendly'.
- The value is assigned to the variable number.

References

- Programming in ANSI C – Balaguruswamy
- https://www.tutorialspoint.com/what-are-the-c-library-functions

Thank You !!!!