Identifiers, Keywords and Data Types

These are the most basic concepts in any programming language and are the building blocks any program.

Identifiers

These are user-defined names given to elements of a program such as variables, functions, arrays, structures, constants etc.

- `int marks` : Identifier `marks` used for variable
- `void display()` : Identifier `display()` used for function
- `int arr[5]` : Identifier `arr` used for array
- `struct Student {...}` : Identifier `Student` used for structures
- `const float PI = 3.14` : Identifier `PI` used for constants

There are rules to create identifiers:

- Identifiers must only contain alphabets (A-Z and a-z), digits (0 - 9), underscore (_)
- They cannot start with a digit

And it is generally adviced to create short and meaningful identifier names in order use and understand them effectively.

Keywords

Keywords are reserved words in a language that comes with special meanings and usage in prgramming languages. They cannot be used anywhere outside their usage scope, otherwise, there will be a syntax or runtime error.

Example: `int`, `float`, `if`, `else`, `char`, `for`, `while`, `const`, `struct` etc.

These keywords form the syntax and grammar of not just `C` language, but in any language. These cannot be used as identifiers.

Data Types

Defines what kind of data a variable should hold. It also decides what operations can be done with it and how much memory it occupies. There are many kinds of data types. Let's look at the basic ones that have been built-in to the language, and does not require a header file to be included.

- `int` stores integer values
- `float` stores decimal numbers upto 6 or 7 decimal places. It occupies 4 bytes of space
- `double` stores decimal numbers upto 15 or 16 places. It occupies 8 bytes of space.

There more variations such as `long int`, `short int`, `unsigned int` etc.

- `char` stores just a single character in ASCII code. You can assign an integer which will b converted to corresponding character according tho the code, or assign a single character in single quotes `''`.
- `void` is a special data type that just says that there is nothing to return. It's literally empty or there's nothing. It is widely used in functions where nothing needs to be returned.

These were the fundamental types. Now there are more which are defined by the user and built from these fundamental types.

- Derived types
  - Arrays: `int a[5]` creates an array (a list) `a` which stores 5 integers in it
  - Pointer: `int *p` creates a pointer variable `p` which stores the address of another variable of type `int`
  - Function: `int add(int, int)` creates a function `add` which accepts two integers as inputs and returns an integer value.
- User-defined types
  - `struct` is a type that containes multiple variables of various types. Usually used to define properties of something.
  - `union` is a memory sharing structure.
  - `enum` are named constants.

- `typedef` gives a new name to an existing type.

Signed and Unsigned Integers

As we saw previously, an integer can be either signed or unsigned. What that means is the variable can store negative values based on what we choose. But the range of storing values changes.

Signed Integer

Being default in C language, it can store values of both negative and positive integers. It can store a range of integers from -2^{n-1} to $2^{n-1} - 1$ where n is the number of bits the data type holds.

Unsigned Integer

It can store values of only positive integers. But it can hold larger numbers than signed integer can. It can store a range of integers from 0 to $2^n - 1$.

This happens because in case of signed integers, the most significant bit (the bit at the left-most end) is used to define the sign of the integer, whereas in unsigned integers, it is a part of the integer.

There are various combinations in which integers can be assigned along with signed and unsigned. They all vary in how much space they occupy, and hence, the range of numbers they can store. Let's see all such types:

Data Type	Size	Range
`short int`	2 bytes	$-32,768$ to $32,767$
`unsigned short int`	2 bytes	0 to $65,535$
`int`	4 bytes	$-2,147,483,648$ to $2,147,483,647$
`unsigned int`	4 bytes	0 to $4,294,967,295$
`long int`	4 bytes (in 32 bit system) or 8 bytes	Same as `int` if 4 bytes and `long long int` if 8 bytes
`unsigned long int`	4 bytes (in 32 bit system) or 8 bytes	Same as `int` if 4 bytes and `long long int` if 8 bytes
`long long int`	8 bytes	$-9,223,372,036,854,775,808$ to $9,223,372,036,854,775,807$

Data Type	Size	Range
`unsigned long long int`	8 bytes	0 to $18,446,744,073,709,551,615$

Common Errors

Errors with keywords:

- `int if = 1;` → `if` is a keyword and cannot be assigned
- `Int i;` → keywords are case-sensitive. `Int` should become `int`

Errors with identifiers:

- `float 1value;` → Identifiers cannot start with digits
- `int pricein$;` → Identifiers must not contain any special symbol other than `_`
- `char thisIsACharUsedForAnExample;` → No error, but is unnessacerily long and reduces readability
- `void ;` → Identifier missing

Errors with data types

- `int pi = 3.14;` → `pi` being an integer ignores decimal values which is data loss
- `printf("%d", 3.14);` → A float value trying to fit into an integer specifier
- `unsigned int x = -12;` → Stores a much larger positive value