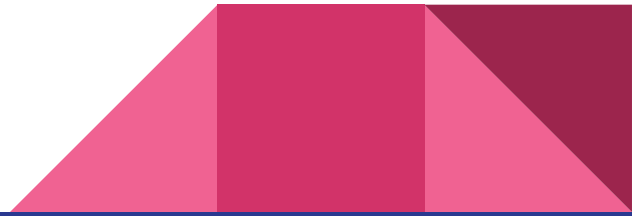
Variables and Identifiers(tokens)

Tokens

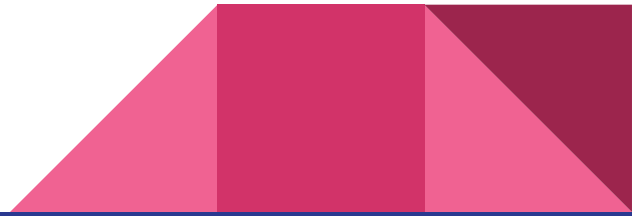
A token is the smallest element of a program that is meaningful to the compiler. Tokens can be classified as follows:

1. Keywords
2. Identifiers
3. Constants
4. Strings
5. Special Symbols
6. Operators



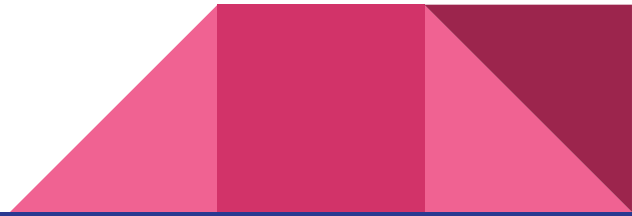
Identifiers

- Identifiers are used for the naming of variables, functions, and arrays.
- It is a string of alphanumeric characters that begins with an alphabet, or an underscore(_) that are used for variables, functions, arrays, structures, unions, and so on.
- It is also known as the user-defined word. Identifier names must differ in spelling and case from any keywords.
- We cannot use keywords as identifiers; they are reserved for special use.
- Once an identifier is declared, we can use the identifier anywhere in the program to refer to the associated value.



Variables

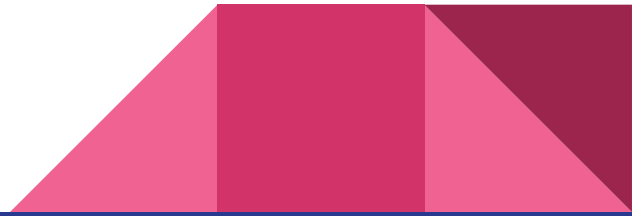
- A variable is a name that points to a memory location.
- It is the basic unit of storage in a program.
- The value of a variable can be changed during program execution.
- All the operations are done on the variable effects that memory location.
- In C, all the variables must be declared before use and in C++ we can declare anywhere in the program at our convenience.



Detailed explaination

A **variable in C** is a memory location associated with some name in order to store some form of data and retrieve it when required. We can store different types of data in the variable and reuse the same variable for storing some other data any number of times.

For using a variable in C, we have to first define it to tell the compiler about its existence so that compiler can allocate the required memory to it.



3 aspects to define a variable

There are 3 aspects of defining a variable:

Variable Declaration

Variable declaration in C tells the compiler about the existence of the variable with the given name and data type. No memory is allocated to a variable in the declaration.

Variable Definition

In the definition of a C variable, the compiler allocates some memory and some value to it. A defined variable will contain some random garbage value till it is not initialized.

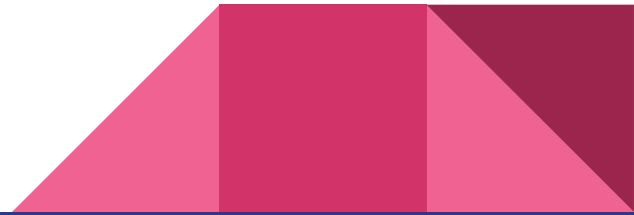
Variable Initialization

Initialization of a variable is the process where the user assigns some meaningful value to the variable.

Rules for naming a variable

Assign any name to the variable as long as it follows the following rules:

1. A variable name must only contain alphabets, digits, and underscore.
2. A variable name must start with an alphabet or an underscore only. It cannot start with a digit.
3. No whitespace is allowed within the variable name.
4. A variable name must not be any reserved word or keyword.

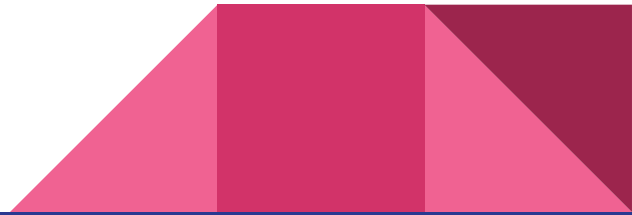


Scope of variables

The **scope of a variable** is the region in which the variable exists it is valid to perform operations on it. Beyond the scope of the variable, we cannot access it and it is said to be out of scope.

On the basis of scope, C variables can be classified into two types:

1. **Local Variables**
2. **Global Variables**



Basic Input Output in C - Characters

- C language has standard libraries that allow input and output in a program. The **stdio.h** or **standard input output library** in C that has methods for input and output.

scanf()

- The scanf() method, in C, reads the value from the console as per the type specified.
- **scanf("%X", &variableOfXType);** where **%X** is the format specifier in C. It is a way to tell the compiler what type of data is in a variable and **&** is the address operator in C, which tells the compiler to change the real value of this variable, stored at this address in the memory.

printf()

- The printf() method, in C, prints the value passed as the parameter to it, on the console screen. **Syntax:**
- **printf("%X", variableOfXType);** where **%X** is the format specifier in C. It is a way to tell the compiler what type of data is in a variable and **&** is the address operator in C, which tells the compiler to change the real value of this variable, stored at this address in the memory.
-

Basic Input Output in C - Formatted IO

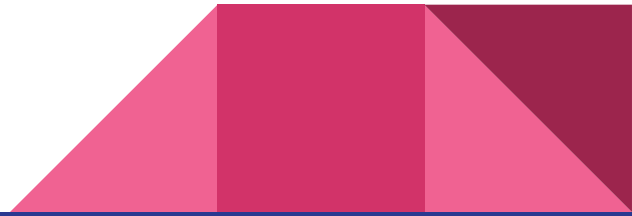
Formatted IO functions are used to take various inputs from the user and display multiple outputs to the user. These types of I/O functions can help to display the output to the user in different formats using the format specifiers. These I/O supports all data types like int, float, char, and many more.

Why they are called formatted I/O?

These functions are called formatted I/O functions because we can use format specifiers in these functions and hence, we can format these functions according to our needs.

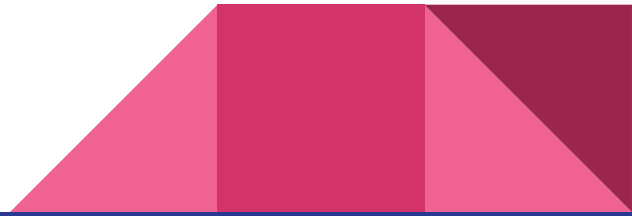
A few interesting formatted functions are:

- printf
- scanf
- sprintf
- sscanf



Type conversion

- Type conversion in C is the process of converting one data type to another.
- The type conversion is only performed to those data types where conversion is possible.
- Type conversion is performed by a compiler.
- In type conversion, the destination data type can’t be smaller than the source data type.
- Type conversion is done at compile time and it is **also called widening conversion** because the destination data type can’t be smaller than the source data type.
- It is of two types:
  - Implicit Conversion
  - Explicit Conversion

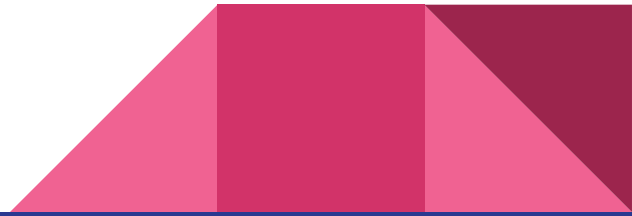


Implicit Conversion

- It is also known as **‘automatic type conversion’**.
- Done by the compiler on its own, without any external trigger from the user.
- Generally takes place when in an expression more than one data type is present. In such conditions type conversion (type promotion) takes place to avoid loss of data.
- All the data types of the variables are upgraded to the data type of the variable with the largest data type.
    - bool -> char -> short int -> int ->
    - unsigned int -> long -> unsigned ->
    - long long -> float -> double -> long double
- It is possible for implicit conversions to lose information, signs can be lost (when signed is implicitly converted to unsigned), and overflow can occur (when long long is implicitly converted to float).

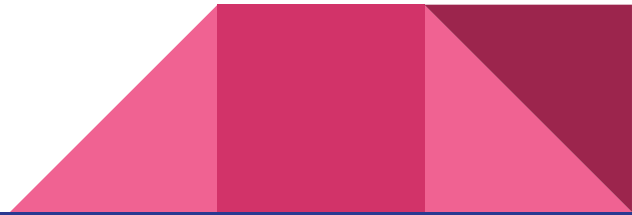
Explicit Conversion

- This process is also called type casting and it is user-defined.
- Here the user can typecast the result to make it of a particular data type.
- Syntax : **(type) expression**



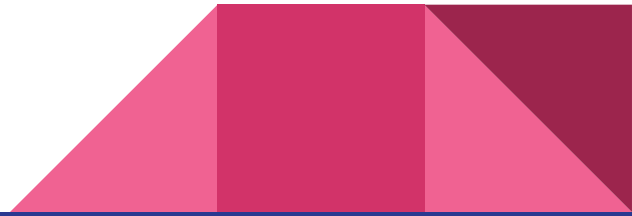
Advantages of Type Conversion

- This is done to take advantage of certain features of type hierarchies or type representations.
- It helps us to compute expressions containing variables of different data types.
- The accuracy of our result increases with the help of type conversion.
- Before performing operations, we can refer to the conversion rank hierarchy to get better results.



Type Casting

- Typecasting in C is the process of converting one data type to another data type by the programmer using the casting operator during program design.
- In typecasting, the destination data type may be smaller than the source data type when converting the data type to another data type, that’s why it is **also called narrowing conversion**.

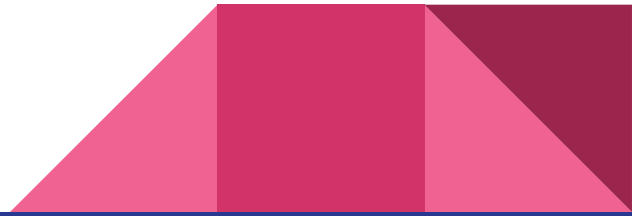


Implicit Type Casting

- Implicit type casting in C is used to convert the data type of any variable without using the actual value that the variable holds.
- It performs the conversions without altering any of the values which are stored in the data variable.
- Conversion of lower data type to higher data type will occur automatically.
- Integer promotion will be performed first by the compiler.
- After that, it will determine whether two of the operands have different data types.

Explicit Type Casting

- There are some cases where if the datatype remains unchanged, it can give incorrect output.
- In such cases, typecasting can help to get the correct output and reduce the time of compilation.
- In explicit type casting, we have to force the conversion between data types.



Advantages of Type Casting

- Type casting in C programming makes the program very lightweight.
- Type representation and hierarchies are some features we can take advantage of with the help of typecasting.
- Type casting helps programmers to convert one data type to another data type.