KNOWLEGDE SUMMARY

Expected, To learn, Extra.

In this document an investigation will be carried out covering some topics interns should know and should learn.

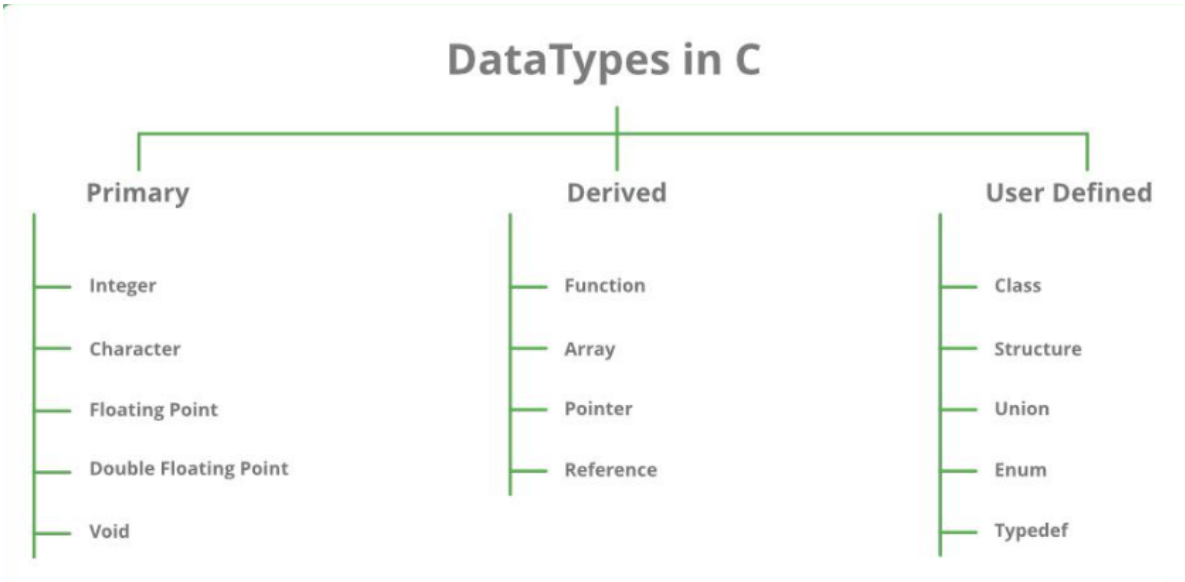
JULIO EMANUEL GONZALEZ GONZALEZ

Expected knowledge.

Data types.

Each variable in C has an associated data type. It specifies the type of data that the variable can store like integer, character, floating, double, etc. Each data type requires different amounts of memory and has some specific operations which can be performed over it. The data type is a collection of data with values having fixed values, meaning as well as its characteristics.

Types.	Description.
Primitive Data Types.	Primitive data types are the most basic data types that are used for representing simple values such as integers, float, characters, etc.
User Defined Data Types.	The user-defined data types are defined by the user himself.
Derived Types.	The data types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types.



Different data types also have different ranges up to which they can store numbers. These ranges may vary from compiler to compiler. Below is a list of ranges along with the memory requirement and format specifiers on the 32-bit GCC compiler.

Data Type	Size (bytes)	Range	Format Specifier
short int	2	-32,768 to 32,767	%hd
unsigned short int	2	0 to 65,535	%hu
unsigned int	4	0 to 4,294,967,295	%u
int	4	-2,147,483,648 to 2,147,483,647	%d
long int	4	-2,147,483,648 to 2,147,483,647	%ld
unsigned long int	4	0 to 4,294,967,295	%lu
long long int	8	-(2^63) to (2^63)-1	%lld
unsigned long long int	8	0 to 18,446,744,073,709,551,615	%llu
signed char	1	-128 to 127	%c
unsigned char	1	0 to 255	%c
float	4	1.2E-38 to 3.4E+38	%f
double	8	1.7E-308 to 1.7E+308	%lf
long double	16	3.4E-4932 to 1.1E+4932	%Lf

The **long, short, signed,** and **unsigned** are datatype modifier that can be used with some primitive data types to change the size or length of the datatype.

Integer Data Type

The integer datatype in C is used to store the integer numbers (any number including positive, negative and zero without decimal part). Octal values, hexadecimal values, and decimal values can be stored in int data type in C.

The integer data type can also be used as

- **unsigned int:** Unsigned int data type in C is used to store the data values from zero to positive numbers but it can't store negative values like signed int.
- **short int:** It is lesser in size than the int by 2 bytes so can only store values from -32,768 to 32,767.
- **long int:** Larger version of the int datatype so can store values greater than int.
- **unsigned short int:** Similar in relationship with short int as unsigned int with int.

The size of an integer data type is compiler dependent. We can use sizeof operator to check the actual size of any data type.

Character Data Type

Character data type allows its variable to store only a single character. The size of the character is 1 byte. It is the most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.

Float Data Type

In C programming float data type is used to store floating-point values. Float in C is used to store decimal and exponential values. It is used to store decimal numbers (numbers with floating point values) with single precision.

Double Data Type

A Double data type in C is used to store decimal numbers (numbers with floating point values) with double precision. It is used to define numeric values which hold numbers with decimal values in C.

The double data type is basically a precision sort of data type that is capable of holding 64 bits of decimal numbers or floating points. Since double has more precision as compared to that float then it is much more obvious that it occupies twice the memory occupied by the floating-point type. It can easily accommodate about 16 to 17 digits after or before a decimal point.

Void Data Type

The void data type in C is used to specify that no value is present. It does not provide a result value to its caller. It has no values and no operations. It is used to represent nothing. Void is used in multiple ways as function return type, function arguments as void, and pointers to void.

Size of Data Types in C

The size of the data types in C is dependent on the size of the architecture, so we cannot define the universal size of the data types. For that, the C language provides the sizeof() operator to check the size of the data types.

Storage Classes.

C Storage Classes are used to describe the features of a variable/function. These features basically include the scope, visibility, and lifetime which help us to trace the existence of a particular variable during the runtime of a program.

C language uses 4 storage classes, namely:

Auto.

This is the default storage class for all the variables declared inside a function or a block. Hence, the keyword auto is rarely used while writing programs in C language. Auto variables can be only accessed within the block/function they have been declared and not outside them (which defines their scope). Of course, these can be accessed within nested blocks within the parent block/function in which the auto variable was declared.

Extern.

Extern storage class simply tells us that the variable is defined elsewhere and not within the same block where it is used. Basically, the value is assigned to it in a different block and this can be overwritten/changed in a different block as well. So an extern variable is nothing but a global variable initialized with a legal value where it is declared in order to be used elsewhere. It can be accessed within any function/block.

Also, a normal global variable can be made extern as well by placing the ‘extern’ keyword before its declaration/definition in any function/block. This basically signifies that we are not initializing a new variable but instead, we are using/accessing the global variable only. The main purpose of using extern variables is that they can be accessed between two different files which are part of a large program.

Static.

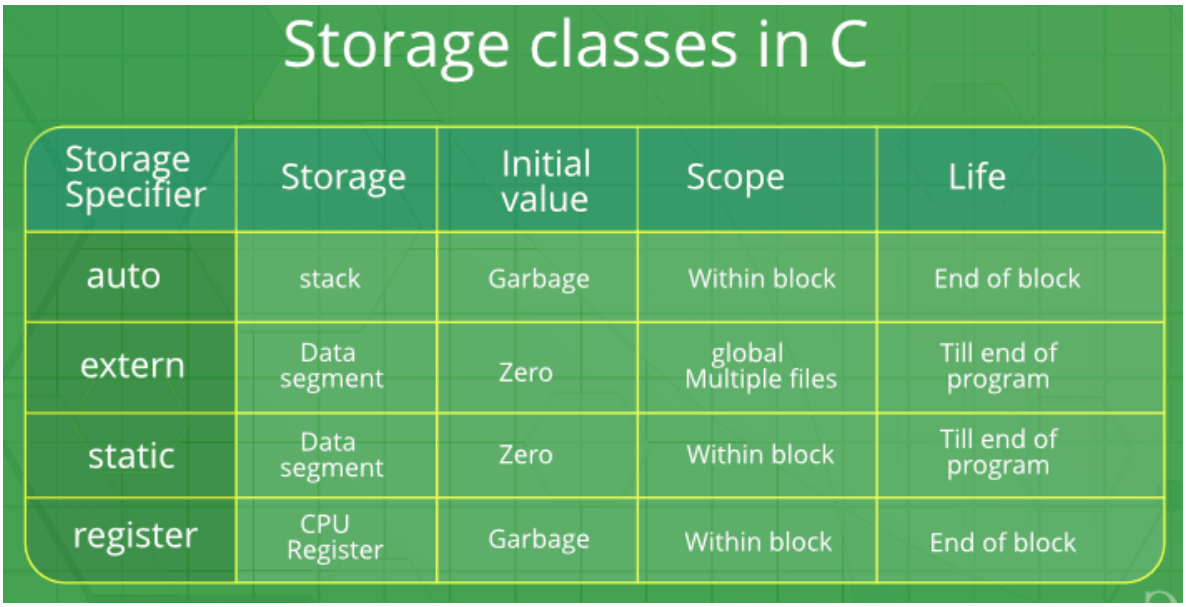
This storage class is used to declare static variables which are popularly used while writing programs in C language. Static variables have the property of preserving their value even after they are out of their scope! Hence, static variables preserve the value of their last use in their scope. So, we can say that they are initialized only once and exist till the termination of the program. Thus, no new memory is allocated because they are not re-declared.

Their scope is local to the function to which they were defined. Global static variables can be accessed anywhere in the program. By default, they are assigned the value 0 by the compiler.

Register.

This storage class declares register variables that have the same functionality as that of the auto variables. The only difference is that the compiler tries to store these variables in the register of the microprocessor if a free register is available. This makes the use of register variables to be much faster than that of the variables stored in the memory during the runtime of the program.

If a free registration is not available, these are then stored in the memory only. Usually, a few variables which are to be accessed very frequently in a program are declared with the register keyword which improves the running time of the program. An important and interesting point to be noted here is that we cannot obtain the address of a register variable using pointers.



Functions.

A function in C is a set of statements that when called perform some specific tasks. It is the basic building block of a C program that provides modularity and code reusability. The programming statements of a function are enclosed within {} braces, having certain meanings and performing certain operations. They are also called subroutines or procedures in other languages.

In this article, we will learn about functions, function definition. declaration, arguments, and parameters, return values, and many more.

Syntax of Functions in C

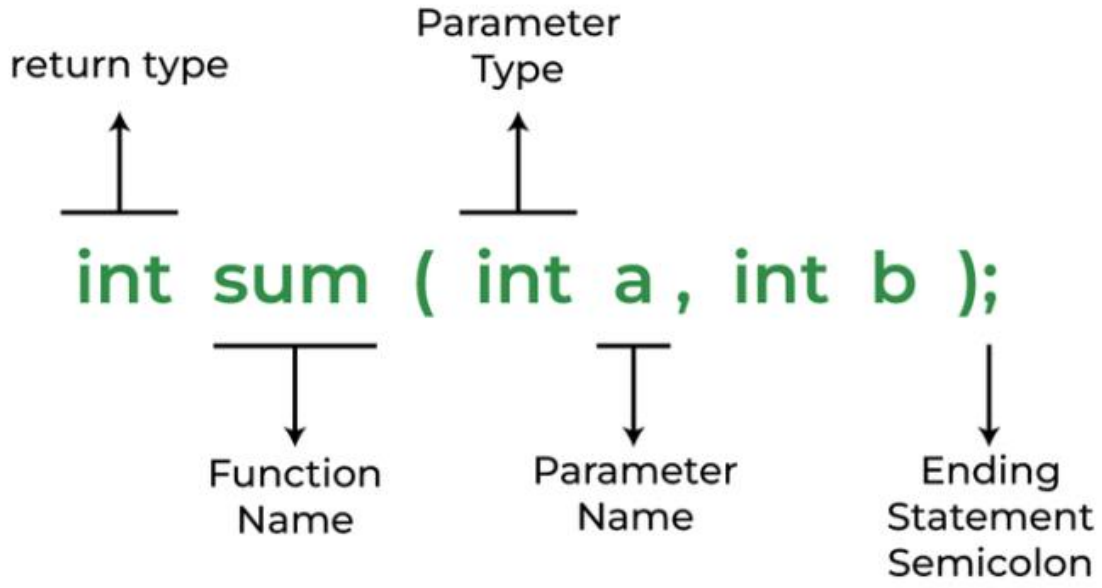
The syntax of function can be divided into 3 aspects:

- Function Declaration
- Function Definition
- Function Calls

Function Declarations.

In a function declaration, we must provide the function name, its return type, and the number and type of its parameters. A function declaration tells the compiler that there is a function with the given name defined somewhere else in the program.

The parameter name is not mandatory while declaring functions. We can also declare the function without using the name of the data variables.

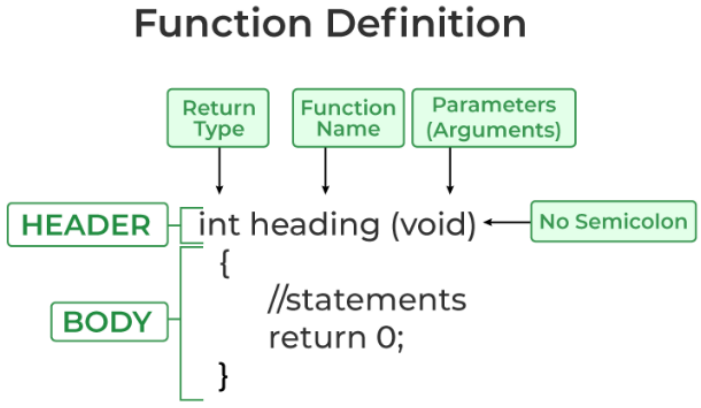


A function in C must always be declared globally before calling it.

Function Definition.

The function definition consists of actual statements which are executed when the function is called (i.e. when the program control comes to the function).

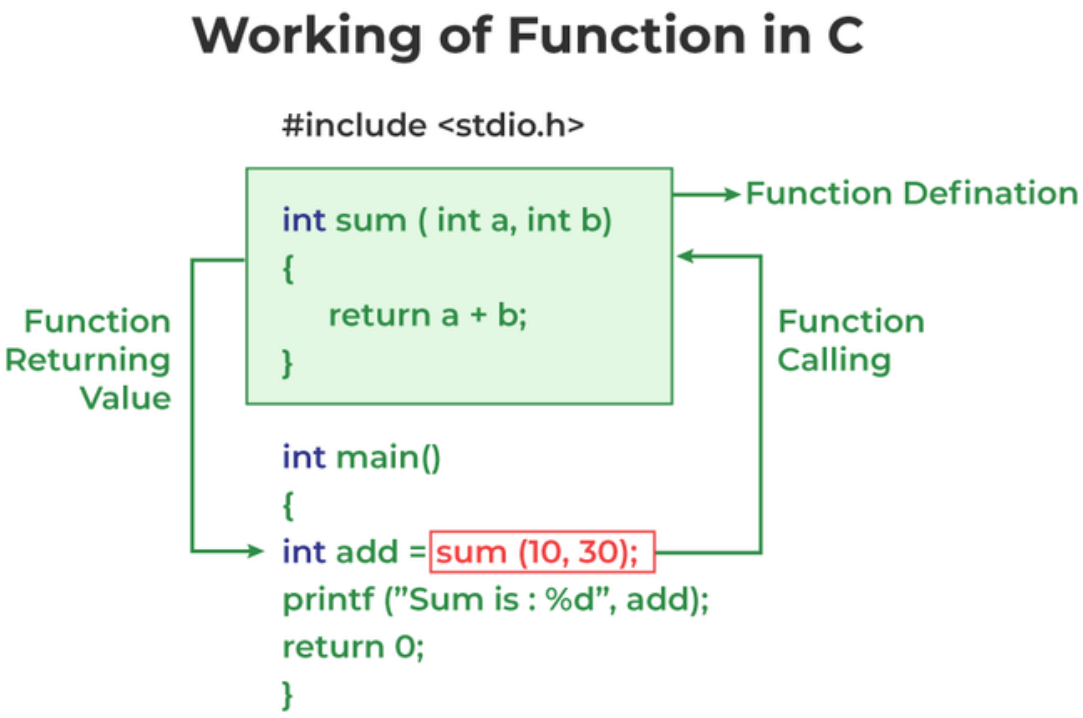
A C function is generally defined and declared in a single step because the function definition always starts with the function declaration, so we do not need to declare it explicitly. The below example serves as both a function definition and a declaration.



Function Call.

A function call is a statement that instructs the compiler to execute the function. We use the function name and parameters in the function call.

In the below example, the first sum function is called and 10,30 are passed to the sum function. After the function call sum of a and b is returned and control is also returned back to the main function of the program.



Function call is necessary to bring the program control to the function definition. If not called, the function statements will not be executed.

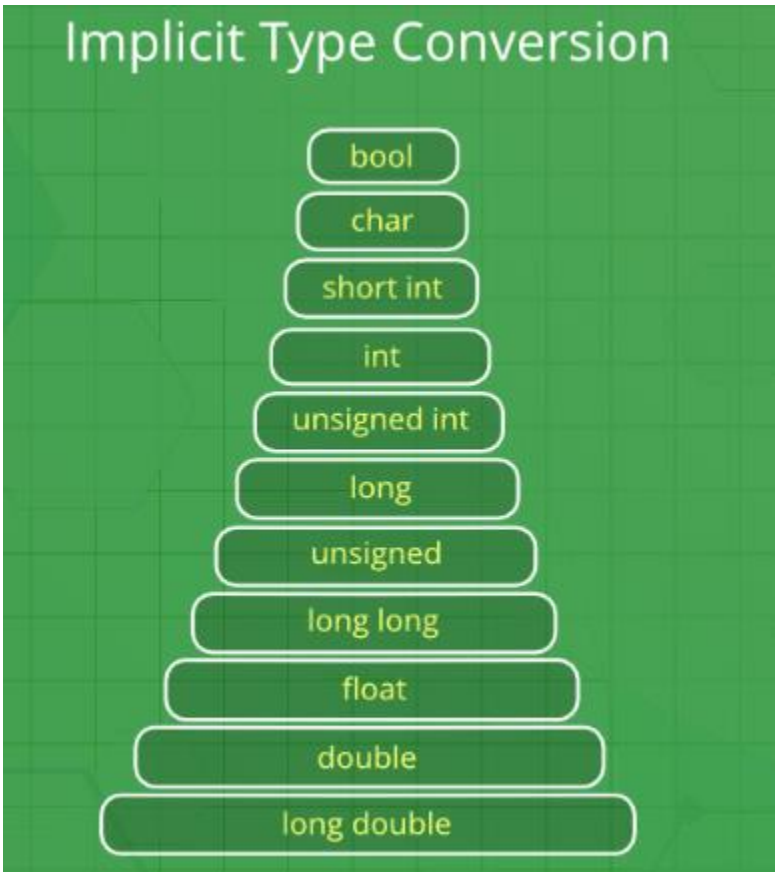
Data type manipulation.

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. There are two types of Conversion:

Implicit Type Conversion.



Also known as 'automatic type conversion'.

A. Done by the compiler on its own, without any external trigger from the user.

B. 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.

C. 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
```

D. 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 is implicitly converted to float).

Occurrences of Implicit Type Conversion in C

Implicit type conversion is also called automatic type conversion. Some of its few occurrences are mentioned below:

- Conversion Rank
- Conversions in Assignment Expressions
- Conversion in other Binary Expressions
- Promotion
- Demotion

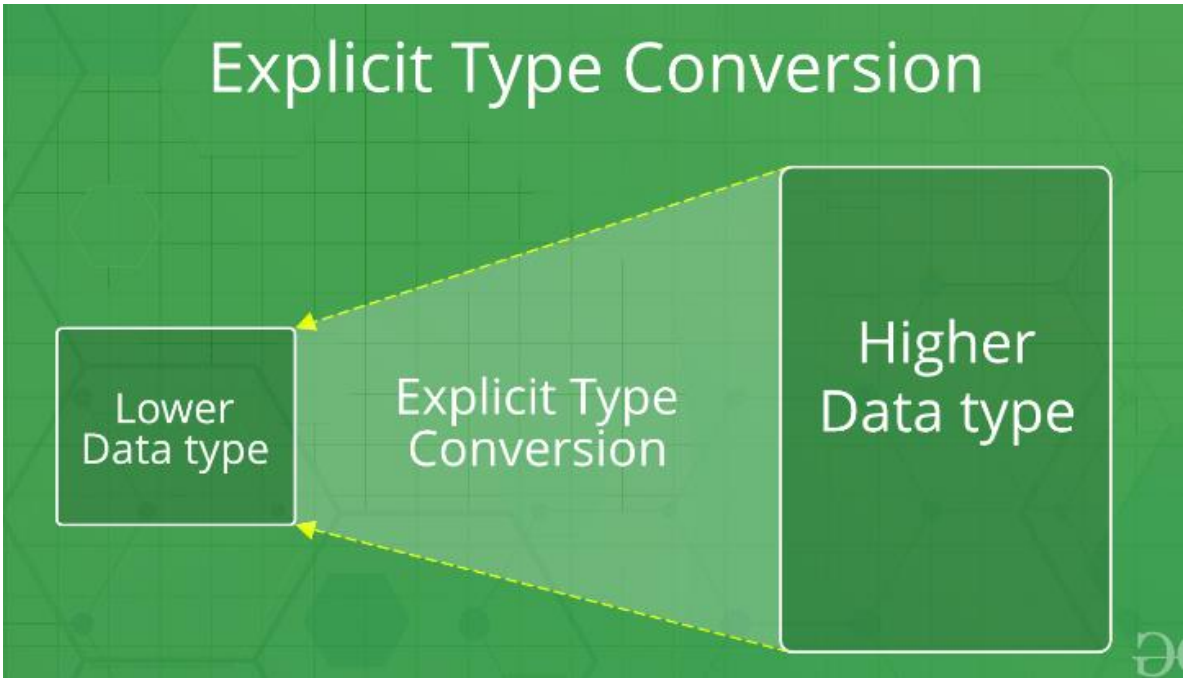
Example.

```c
// An example of implicit conversion
#include <stdio.h>
int main()
{
    int x = 10; // integer x
    char y = 'a'; // character c

    // y implicitly converted to int. ASCII
    // value of 'a' is 97
    x = x + y;

    // x is implicitly converted to float
    float z = x + 1.0;

    printf("x = %d, z = %f", x, z);
    return 0;
}
```

Output

```
x = 107, z = 108.000000
```

Explicit Type 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. The syntax in C Programming:

```
(type) expression
```

Type indicated the data type to which the final result is converted.

Example.

```c
// C program to demonstrate explicit type casting
#include<stdio.h>

int main()
{
    double x = 1.2;

    // Explicit conversion from double to int
    int sum = (int)x + 1;

    printf("sum = %d", sum);

    return 0;
}
```

Output

```
sum = 2
```

Typecasting.

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.

Syntax:

```
int x;
float y;
y = (float) x;
```

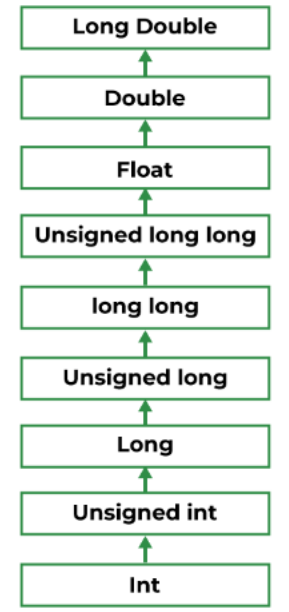
Types of Type Casting.

In C there are two major types to perform type casting.

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. Using the hierarchy below, the conversion would appear as follows if they both have varied 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. This type of casting is explicitly defined within the program.

```c
// C program to illustrate the use of
// typecasting
#include <stdio.h>

// Driver Code
int main()
{
    // Given a & b
    int a = 15, b = 2;
    float div;

    // Division of a and b
    div = a / b;

    printf("The result is %f\n", div);

    return 0;
}
```

```c
// C program to showcase the use of
// typecasting
#include <stdio.h>

// Driver Code
int main()
{
    // Given a & b
    int a = 15, b = 2;
    char x = 'a';

    double div;

    // Explicit Typecasting in double
    div = (double)a / b;

    // converting x implicitly to a+3 i.e, a+3 = d
    x = x + 3;

    printf("The result of Implicit typecasting is %c\n", x);

    printf("The result of Explicit typecasting is %f", div);

    return 0;
}
```

Output

```
The result of Implicit typecasting is d
The result of Explicit typecasting is 7.500000
```

Explanation: In the above C program, the expression (double) converts variable a from type int to type double before the operation.

In C programming, there are 5 built-in type casting functions.

atof(): This function is used for converting the string data type into a float data type.

atbol(): This function is used for converting the string data type into a long data type.

Itoa(): This function is used to convert the long data type into the string data type.

itoba(): This function is used to convert an int data type into a string data type.

atoi(): This data type is used to convert the string data type into an int data type.

Difference between Type Casting and Type Conversion.

TYPE CASTING	TYPE CONVERSION
In type casting, a data type is converted into another data type by a programmer using casting operator.	Whereas in type conversion, a data type is converted into another data type by a compiler.
Type casting can be applied to compatible data types as well as incompatible data types.	Whereas type conversion can only be applied to compatible datatypes.
In type casting, casting operator is needed in order to cast a data type to another data type.	Whereas in type conversion, there is no need for a casting operator.
In typing casting, the destination data type may be smaller than the source data type, when converting the data type to another data type.	Whereas in type conversion, the destination data type can't be smaller than source data type.

Type casting takes place during the program design by programmer.	Whereas type conversion is done at the compile time.
Type casting is also called narrowing conversion because in this, the destination data type may be smaller than the source data type.	Whereas type conversion is also called widening conversion because in this, the destination data type cannot be smaller than the source data type.
Type casting is often used in coding and competitive programming works.	Whereas type conversion is less used in coding and competitive programming as it might cause incorrect answer.

Pointers.

Pointers are one of the core components of the C programming language. A pointer can be used to store the memory address of other variables, functions, or even other pointers. The use of pointers allows low-level memory access, dynamic memory allocation, and many other functionalities in C.

In this article, we will discuss C pointers in detail, their types, uses, advantages, and disadvantages with examples.

What is a Pointer in C?

A pointer is defined as a derived data type that can store the address of other C variables or a memory location. We can access and manipulate the data stored in that memory location using pointers.

As the pointers in C store the memory addresses, their size is independent of the type of data they are pointing to. This size of pointers in C only depends on the system architecture.

Syntax of C Pointers.

The syntax of pointers is similar to the variable declaration in C, but we use the (*) dereferencing operator in the pointer declaration.

```
datatype * ptr;
```

Where,

- ptr is the name of the pointer.
- datatype is the type of data it is pointing to.

The above syntax is used to define a pointer to a variable. We can also define pointers to functions, structures, etc.

How to Use Pointers?

The use of pointers in C can be divided into three steps:

1. Pointer Declaration
2. Pointer Initialization
3. Pointer Dereferencing

Pointer Declaration.

In pointer declaration, we only declare the pointer but do not initialize it. To declare a pointer, we use the (*) dereference operator before its name.

Example

```
int *ptr;
```

The pointer declared here will point to some random memory address as it is not initialized. Such pointers are called wild pointers.

Pointer Initialization.

Pointer initialization is the process where we assign some initial value to the pointer variable. We generally use the (&) address of operator to get the memory address of a variable and then store it in the pointer variable.

Example

```
int var = 10;
int * ptr;
ptr = &var;
```

We can also declare and initialize the pointer in a single step. This method is called pointer definition as the pointer is declared and initialized at the same time.

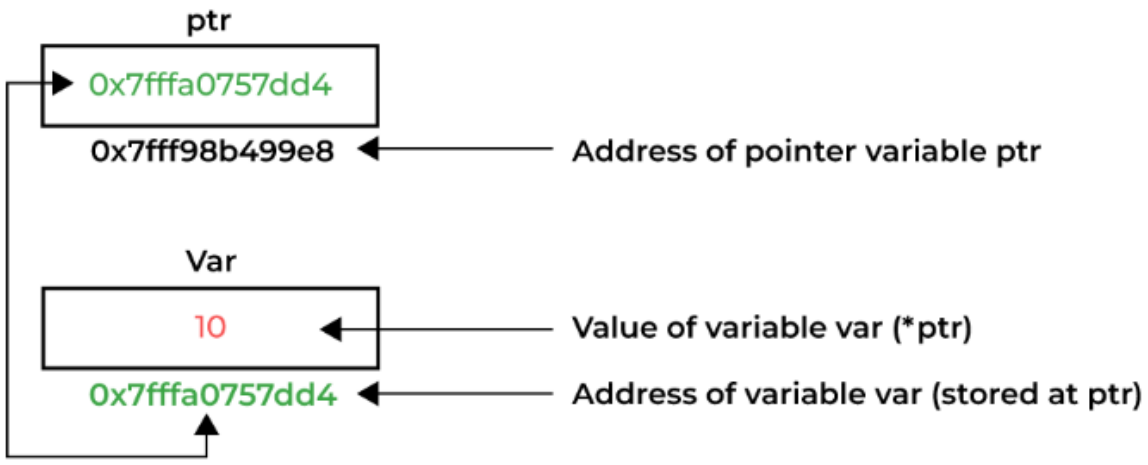
Example

```
int *ptr = &var;
```

It is recommended that the pointers should always be initialized to some value before starting using it. Otherwise, it may lead to number of errors.

Pointer Dereferencing.

Dereferencing a pointer is the process of accessing the value stored in the memory address specified in the pointer. We use the same (*) dereferencing operator that we used in the pointer declaration.



```c
// C program to illustrate Pointers
#include <stdio.h>

void geeks()
{
    int var = 10;

    // declare pointer variable
    int* ptr;

    // note that data type of ptr and var must be same
    ptr = &var;

    // assign the address of a variable to a pointer
    printf("Value at ptr = %p \n", ptr);
    printf("Value at var = %d \n", var);
    printf("Value at *ptr = %d \n", *ptr);
}

// Driver program
int main()
{
    geeks();
    return 0;
}
```

Output

```
Value at ptr = 0x7fff1038675c
Value at var = 10
Value at *ptr = 10
```

Types of Pointers in C

Pointers in C can be classified into many different types based on the parameter on which we are defining their types. If we consider the type of variable stored in the memory location pointed by the pointer, then the pointers can be classified into the following types:

Integer Pointers.

As the name suggests, these are the pointers that point to the integer values.

Syntax.

```
int *ptr;
```

These pointers are pronounced as Pointer to Integer.

Similarly, a pointer can point to any primitive data type. It can point also point to derived data types such as arrays and user-defined data types such as structures.

Array Pointer.

Pointers and Array are closely related to each other. Even the array name is the pointer to its first element. They are also known as Pointer to Arrays. We can create a pointer to an array using the given syntax.

Syntax.

```
char *ptr = &array_name;
```

Pointer to Arrays exhibits some interesting properties which we discussed later in this article.

Structure Pointer.

The pointer pointing to the structure type is called Structure Pointer or Pointer to Structure. It can be declared in the same way as we declare the other primitive data types.

Syntax.

```
struct struct_name *ptr;
```

In C, structure pointers are used in data structures such as linked lists, trees, etc.

Function Pointers.

Function pointers point to the functions. They are different from the rest of the pointers in the sense that instead of pointing to the data, they point to the code. Let's consider a function prototype – int func (int, char), the function pointer for this function will be

Syntax.

```
int (*ptr)(int, char);
```

The syntax of the function pointers changes according to the function prototype.

Double Pointers.

In C language, we can define a pointer that stores the memory address of another pointer. Such pointers are called double-pointers or pointers-to-pointer. Instead of pointing to a data value, they point to another pointer.

Syntax.

```
datatype ** pointer_name;
```

Dereferencing Double Pointer.

```
*pointer_name; // get the address stored in the inner level pointer
**pointer_name; // get the value pointed by inner level pointer
```

In C, we can create multi-level pointers with any number of levels such as – ***ptr3, ****ptr4, ******ptr5 and so on.

NULL Pointer

The Null Pointers are those pointers that do not point to any memory location. They can be created by assigning a NULL value to the pointer. A pointer of any type can be assigned the NULL value.

Syntax.

```
data_type *pointer_name = NULL;
        or
pointer_name = NULL
```

It is said to be good practice to assign NULL to the pointers currently not in use.

Void Pointer.

The Void pointers in C are the pointers of type void. It means that they do not have any associated data type. They are also called generic pointers as they can point to any type and can be typecasted to any type.

Syntax.

```
void * pointer_name;
```

One of the main properties of void pointers is that they cannot be dereferenced.

Wild Pointers.

The Wild Pointers are pointers that have not been initialized with something yet. These types of C-pointers can cause problems in our programs and can eventually cause them to crash.

Example

```
int *ptr;
char *str;
```

Constant Pointers.

In constant pointers, the memory address stored inside the pointer is constant and cannot be modified once it is defined. It will always point to the same memory address.

Syntax.

```
data_type * const pointer_name;
```

Pointer to Constant.

The pointers pointing to a constant value that cannot be modified are called pointers to a constant. Here we can only access the data pointed by the pointer but cannot modify it. Although, we can change the address stored in the pointer to constant.

Syntax.

```
const data_type * pointer_name;
```

Other Types of Pointers in C.

There are also the following types of pointers available to use in C apart from those specified above:

- Far pointer: A far pointer is typically 32-bit that can access memory outside the current segment.
- Dangling pointer: A pointer pointing to a memory location that has been deleted (or freed) is called a dangling pointer.
- Huge pointer: A huge pointer is 32-bit long containing segment address and offset address.
- Complex pointer: Pointers with multiple levels of indirection.
- Near pointer: Near pointer is used to store 16-bit addresses means within the current segment on a 16-bit machine.
- Normalized pointer: It is a 32-bit pointer, which has as much of its value in the segment register as possible.
- File Pointer: The pointer to a FILE data type is called a stream pointer or a file pointer.

Size of Pointers in C

The size of the pointers in C is equal for every pointer type. The size of the pointer does not depend on the type it is pointing to. It only depends on the operating system and CPU architecture. The size of pointers in C is:

- 8 bytes for a 64-bit System
- 4 bytes for a 32-bit System

The reason for the same size is that the pointers store the memory addresses, no matter what type they are. As the space required to store the addresses of the different memory locations is the same, the memory required by one pointer type will be equal to the memory required by other pointer types.

How to find the size of pointers in C?

We can find the size of pointers using the sizeof operator as shown in the following program:

Example: C Program to find the size of different pointer types.

```c
// C Program to find the size of different pointers types
#include <stdio.h>

// dummy structure
struct str {
};

// dummy function
void func(int a, int b){};

int main()
{
    // dummy variables definitions
    int a = 10;
    char c = 'G';
    struct str x;

    // pointer definitions of different types
    int* ptr_int = &a;
    char* ptr_char = &c;
    struct str* ptr_str = &x;
    void (*ptr_func)(int, int) = &func;
    void* ptr_vn = NULL;

    // printing sizes
    printf("Size of Integer Pointer  \t:\t%d bytes\n",
           sizeof(ptr_int));
    printf("Size of Character Pointer\t:\t%d bytes\n",
           sizeof(ptr_char));
    printf("Size of Structure Pointer\t:\t%d bytes\n",
           sizeof(ptr_str));
    printf("Size of Function Pointer\t:\t%d bytes\n",
           sizeof(ptr_func));
    printf("Size of NULL Void Pointer\t:\t%d bytes",
           sizeof(ptr_vn));

    return 0;
}
```

Decision making.

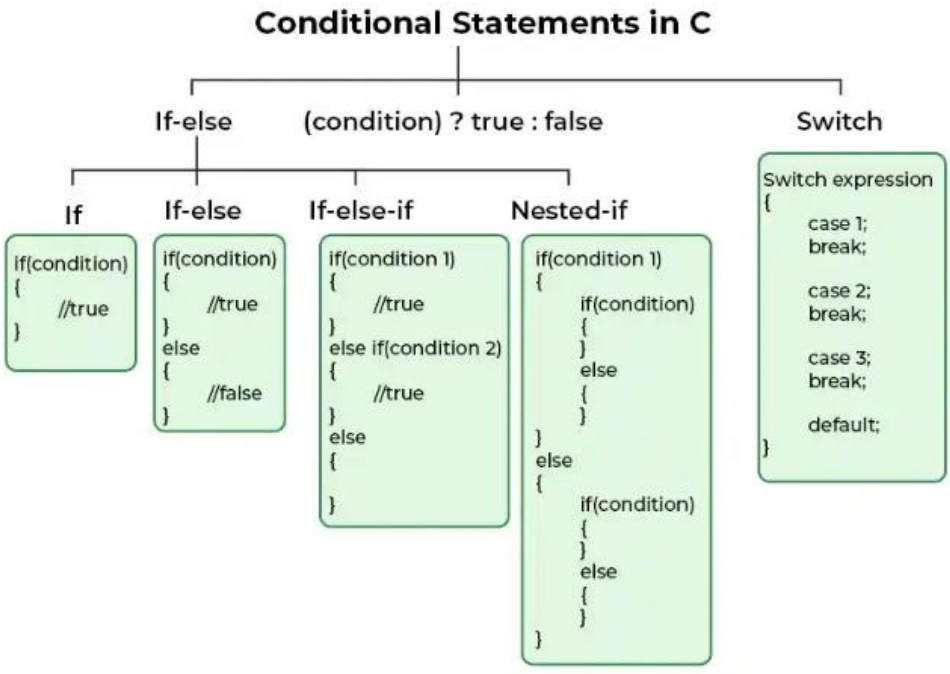
The conditional statements (also known as decision control structures) such as if, if else, switch, etc. are used for decision-making purposes in C programs.

They are also known as Decision-Making Statements and are used to evaluate one or more conditions and make the decision whether to execute a set of statements or not. These decision-making statements in programming languages decide the direction of the flow of program execution.

Need of Conditional Statements.

They're come situations in real life when we need to make some decisions and based on these decisions, we decide what should we do next. Similar situations arise in programming also where we need to make some decisions and based on these decisions, we will execute the next block of code. For example, in C if x occurs then execute y else execute z. There can also be multiple conditions like in C if x occurs then execute p, else if condition y occurs execute q, else execute r. This condition of C else if is one of the many ways of importing multiple conditions.

Types of Conditional Statements in C



Following are the decision-making statements available in C:

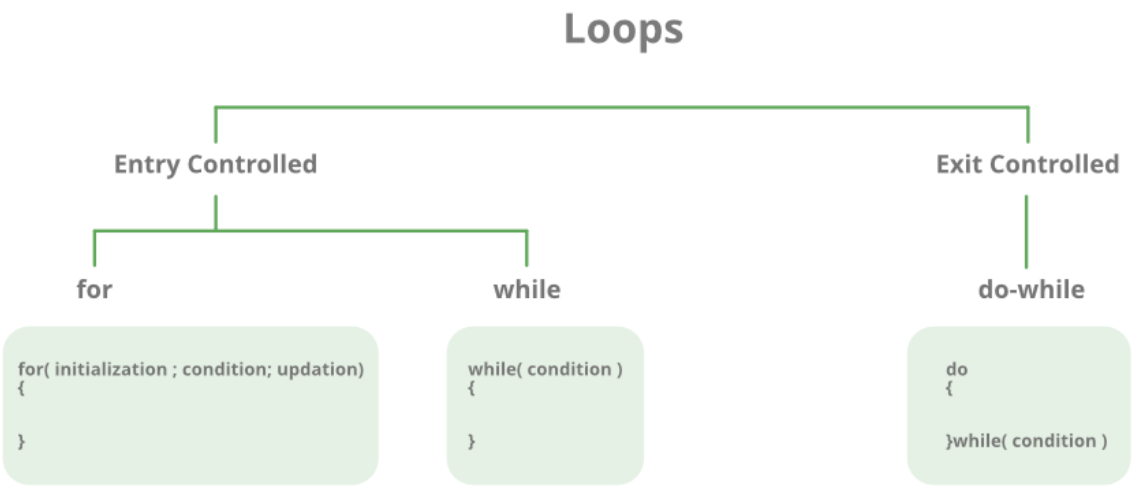
1. if Statement
2. if-else Statement
3. Nested if Statement.
4. if-else-if Ladder
5. Switch Statement
6. Conditional Operator
7. Jump Statements:
    - break
    - continue
    - goto
    - return

Looping.

Loops in programming are used to repeat a block of code until the specified condition is met. A loop statement allows programmers to execute a statement or group of statements multiple times without repetition of code.

There are mainly two types of loops in C Programming:

- Entry Controlled loops: In Entry controlled loops the test condition is checked before entering the main body of the loop. For Loop and While Loop is Entry-controlled loops.
- Exit Controlled loops: In Exit controlled loops the test condition is evaluated at the end of the loop body. The loop body will execute at least once, irrespective of whether the condition is true or false. do-while Loop is Exit Controlled loop.



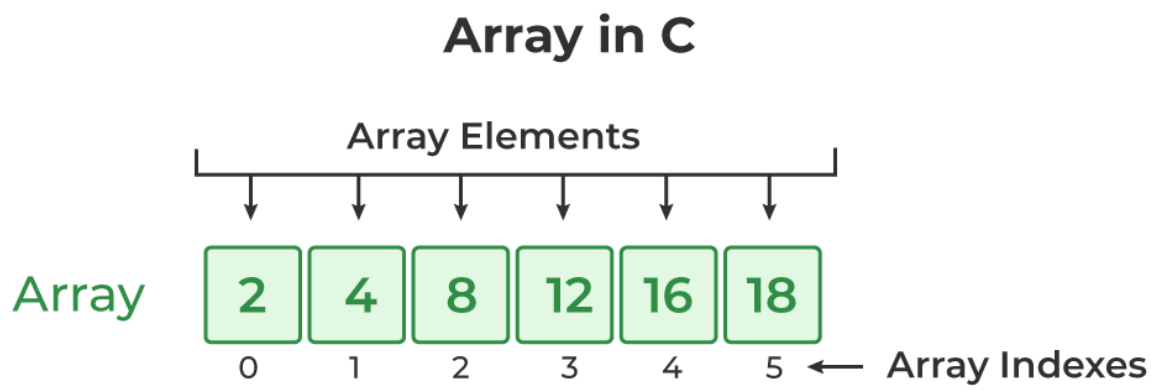
Loop Type.	Description.
for loop	first Initializes, then condition check, then executes the body and at last, the update is done.
while loop	first Initializes, then condition checks, and then executes the body, and updating can be inside the body.
do-while loop	do-while first executes the body and then the condition check is done.

Arrays.

Array in C is one of the most used data structures in C programming. It is a simple and fast way of storing multiple values under a single name. In this article, we will study the different aspects of array in C language such as array declaration, definition, initialization, types of arrays, array syntax, advantages and disadvantages, and many more.

What is Array in C?

An array in C is a fixed-size collection of similar data items stored in contiguous memory locations. It can be used to store the collection of primitive data types such as int, char, float, etc., and also derived and user-defined data types such as pointers, structures, etc



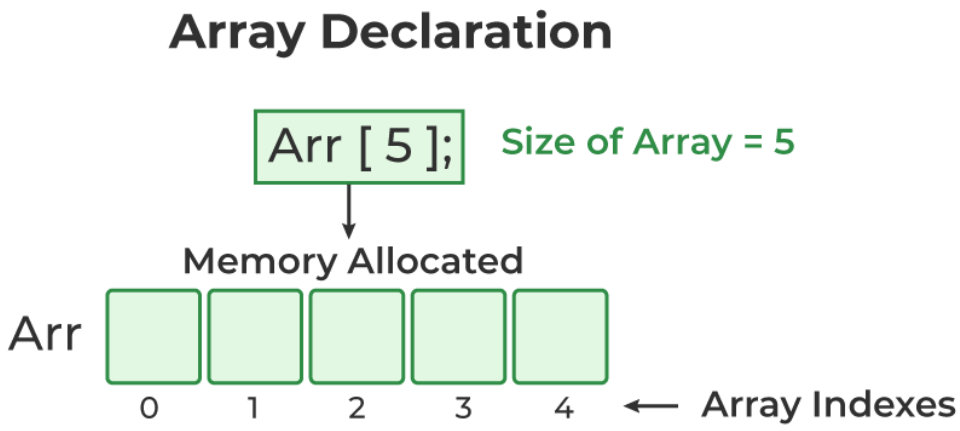
C Array Declaration.

In C, we have to declare the array like any other variable before using it. We can declare an array by specifying its name, the type of its elements, and the size of its dimensions. When we declare an array in C, the compiler allocates the memory block of the specified size to the array name.

Syntax of Array Declaration.

```
data_type array_name [size];
        or
data_type array_name [size1] [size2]...[sizeN];
```

where N is the number of dimensions.



The C arrays are static in nature, i.e., they are allocated memory at the compile time.

Example of Array Declaration.

```c
// C Program to illustrate the array declaration
#include <stdio.h>

int main()
{

    // declaring array of integers
    int arr_int[5];
    // declaring array of characters
    char arr_char[5];

    return 0;
}
```

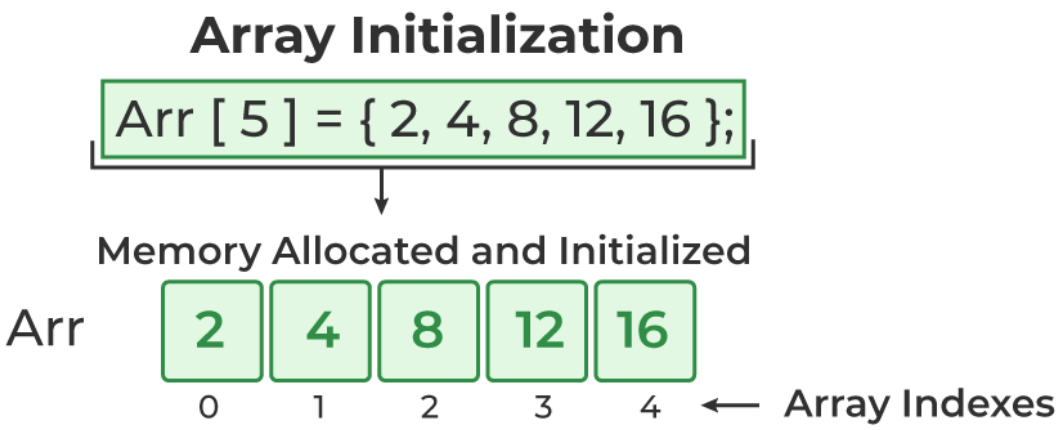
C Array Initialization.

Initialization in C is the process to assign some initial value to the variable. When the array is declared or allocated memory, the elements of the array contain some garbage value. So, we need to initialize the array to some meaningful value. There are multiple ways in which we can initialize an array in C.

Array Initialization with Declaration.

In this method, we initialize the array along with its declaration. We use an initializer list to initialize multiple elements of the array. An initializer list is the list of values enclosed within braces {} separated by a comma.

```
data_type array_name [size] = {value1, value2, ... valueN};
```



Array Initialization with Declaration without Size.

If we initialize an array using an initializer list, we can skip declaring the size of the array as the compiler can automatically deduce the size of the array in these cases. The size of the array in these cases is equal to the number of elements present in the initializer list as the compiler can automatically deduce the size of the array.

```
data_type array_name[] = {1,2,3,4,5};
```

The size of the above arrays is 5 which is automatically deduced by the compiler.

Array Initialization after Declaration (Using Loops)

We initialize the array after the declaration by assigning the initial value to each element individually. We can use for loop, while loop, or do-while loop to assign the value to each element of the array.

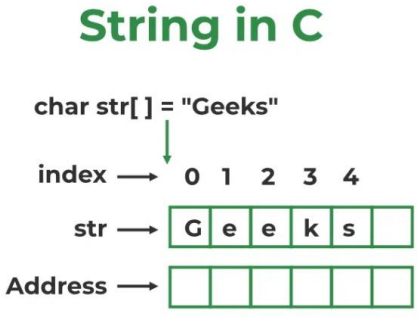
```
for (int i = 0; i < N; i++) {
    array_name[i] = valuei;
}
```

Example of Array Initialization in C.

```c
// C Program to demonstrate array initialization
#include <stdio.h>

int main()
{

    // array initialization using initialier list
    int arr[5] = { 10, 20, 30, 40, 50 };

    // array initialization using initializer list without
    // specifying size
    int arr1[] = { 1, 2, 3, 4, 5 };

    // array initialization using for loop
    float arr2[5];
    for (int i = 0; i < 5; i++) {
        arr2[i] = (float)i * 2.1;
    }
    return 0;
}
```

Strings.

A String in C programming is a sequence of characters terminated with a null character '\0'. The C String is stored as an array of characters. The difference between a character array and a C string is that the string in C is terminated with a unique character '\0'.



C String Declaration Syntax.

Declaring a string in C is as simple as declaring a one-dimensional array. Below is the basic syntax for declaring a string.

```
char string_name[size];
```

In the above syntax string_name is any name given to the string variable and size is used to define the length of the string, i.e the number of characters strings will store.

There is an extra terminating character which is the Null character ('\0') used to indicate the termination of a string that differs strings from normal character arrays.

Example.

```
// C Program to take input string which is separated by
// whitespaces
#include <stdio.h>

// driver code
int main()
{

    char str[20];
    // taking input string
    scanf("%s", str);

    // printing the read string
    printf("%s", str);

    return 0;
}
```

Macros.

In C, a macro is a piece of code in a program that is replaced by the value of the macro. Macro is defined by #define directive. Whenever a macro name is encountered by the

compiler, it replaces the name with the definition of the macro. Macro definitions need not be terminated by a semi-colon (;).

Examples of Macros in C

Below are the programs to illustrate the use of macros in C:

Example 1

The below example demonstrates the use of macros to define LIMIT.

```c
// C program to illustrate macros
#include <stdio.h>

// Macro definition
#define LIMIT 5

// Driver Code
int main()
{
    // Print the value of macro defined
    printf("The value of LIMIT"
           " is %d",
           LIMIT);

    return 0;
}
```

OUTPUT.

```
The value of LIMIT is 5
```

Example 2

The below example demonstrates the use of macros to find the area of a rectangle.

```c
// C program to illustrate macros
#include <stdio.h>

// Macro definition
#define AREA(l, b) (l * b)

// Driver Code
int main()
{
    // Given lengths l1 and l2
    int l1 = 10, l2 = 5, area;

    // Find the area using macros
    area = AREA(l1, l2);

    // Print the area
    printf("Area of rectangle"
           " is: %d",
           area);

    return 0;
}
```

Output

```
Area of rectangle is: 50
```

Explanation

From the above program, we can see that whenever the compiler finds AREA (l, b) in the program it replaces it with the macros definition i.e., (l*b). The values passed to the macro template AREA (l, b) will also be replaced by the statement (l*b). Therefore, AREA (10, 5) will be equal to 10*5.

C Structures (structs).

Structures

Structures (also called structs) are a way to group several related variables into one place. Each variable in the structure is known as a member of the structure.

Unlike an array, a structure can contain many different data types (int, float, char, etc.).

Create a Structure

You can create a structure by using the struct keyword and declare each of its members inside curly braces:

```
struct MyStructure {   // Structure declaration
  int myNum;           // Member (int variable)
  char myLetter;       // Member (char variable)
}; // End the structure with a semicolon
```

To access the structure, you must create a variable of it.

Use the struct keyword inside the main () method, followed by the name of the structure and then the name of the structure variable:

Create a struct variable with the name "s1":

```
struct myStructure {
  int myNum;
  char myLetter;
};

int main() {
  struct myStructure s1;
  return 0;
}
```

Access Structure Members

To access members of a structure, use the dot syntax (.):

Example

```c
// Create a structure called myStructure
struct myStructure {
  int myNum;
  char myLetter;
};

int main() {
  // Create a structure variable of myStructure called s1
  struct myStructure s1;

  // Assign values to members of s1
  s1.myNum = 13;
  s1.myLetter = 'B';

  // Print values
  printf("My number: %d\n", s1.myNum);
  printf("My letter: %c\n", s1.myLetter);

  return 0;
}
```

Now you can easily create multiple structure variables with different values, using just one structure:

Example

```c
// Create different struct variables
struct myStructure s1;
struct myStructure s2;

// Assign values to different struct variables
s1.myNum = 13;
s1.myLetter = 'B';

s2.myNum = 20;
s2.myLetter = 'C';
```

What About Strings in Structures?

Remember that strings in C are actually an array of characters, and unfortunately, you can't assign a value to an array like this:

Example

```
struct myStructure {
  int myNum;
  char myLetter;
  char myString[30];  // String
};

int main() {
  struct myStructure s1;

  // Trying to assign a value to the string
  s1.myString = "Some text";

  // Trying to print the value
  printf("My string: %s", s1.myString);

  return 0;
}
```

An error will occur:

```
prog.c:12:15: error: assignment to expression with array type
```

However, there is a solution for this! You can use the strcpy() function and assign the value to s1.myString, like this:

Example

```c
struct myStructure {
  int myNum;
  char myLetter;
  char myString[30]; // String
};

int main() {
  struct myStructure s1;

  // Assign a value to the string using the strcpy function
  strcpy(s1.myString, "Some text");

  // Print the value
  printf("My string: %s", s1.myString);

  return 0;
}
```

Result:

```
My string: Some text
```

Simpler Syntax

You can also assign values to members of a structure variable at declaration time, in a single line.

Just insert the values in a comma-separated list inside curly braces {}. Note that you don't have to use the strcpy() function for string values with this technique:

Example

```c
// Create a structure
struct myStructure {
  int myNum;
  char myLetter;
  char myString[30];
};

int main() {
  // Create a structure variable and assign values to it
  struct myStructure s1 = {13, 'B', "Some text"};

  // Print values
  printf("%d %c %s", s1.myNum, s1.myLetter, s1.myString);

  return 0;
}
```

Copy Structures

You can also assign one structure to another.

In the following example, the values of s1 are copied to s2:

Example

```c
struct myStructure s1 = {13, 'B', "Some text"};
struct myStructure s2;

s2 = s1;
```

Modify Values

If you want to change/modify a value, you can use the dot syntax (.).

And to modify a string value, the strcpy() function is useful again:

Example

```
struct myStructure {
  int myNum;
  char myLetter;
  char myString[30];
};

int main() {
  // Create a structure variable and assign values to it
  struct myStructure s1 = {13, 'B', "Some text"};

  // Modify values
  s1.myNum = 30;
  s1.myLetter = 'C';
  strcpy(s1.myString, "Something else");

  // Print values
  printf("%d %c %s", s1.myNum, s1.myLetter, s1.myString);

  return 0;
}
```

Modifying values are especially useful when you copy structure values:

```
// Create a structure variable and assign values to it
struct myStructure s1 = {13, 'B', "Some text"};

// Create another structure variable
struct myStructure s2;

// Copy s1 values to s2
s2 = s1;

// Change s2 values
s2.myNum = 30;
s2.myLetter = 'C';
strcpy(s2.myString, "Something else");

// Print values
printf("%d %c %s\n", s1.myNum, s1.myLetter, s1.myString);
printf("%d %c %s\n", s2.myNum, s2.myLetter, s2.myString);
```

Real-Life Example

Use a structure to store different information about Cars:

Example

```c
struct Car {
  char brand[50];
  char model[50];
  int year;
};

int main() {
  struct Car car1 = {"BMW", "X5", 1999};
  struct Car car2 = {"Ford", "Mustang", 1969};
  struct Car car3 = {"Toyota", "Corolla", 2011};

  printf("%s %s %d\n", car1.brand, car1.model, car1.year);
  printf("%s %s %d\n", car2.brand, car2.model, car2.year);
  printf("%s %s %d\n", car3.brand, car3.model, car3.year);

  return 0;
}
```

Referencias

[1] "Data types in C," GeeksforGeeks, 30-Jun-2015. [Online]. Available: https://www.geeksforgeeks.org/data-types-in-c/. [Accessed: 11-Jan-2024].

[2] "Storage classes in C," GeeksforGeeks, 18-Jul-2015. [Online]. Available: https://www.geeksforgeeks.org/storage-classes-in-c/. [Accessed: 11-Jan-2024].

[3] K. Follow, "C functions," GeeksforGeeks, 09-Oct-2022. [Online]. Available: https://www.geeksforgeeks.org/c-functions/. [Accessed: 11-Jan-2024].

[4] "Type conversion in C," GeeksforGeeks, 04-Apr-2016. [Online]. Available: https://www.geeksforgeeks.org/type-conversion-c/. [Accessed: 11-Jan-2024].

[5] P. prernap1909 Follow, "C- TypeCasting," GeeksforGeeks, 20-Jan-2021. [Online]. Available: https://www.geeksforgeeks.org/c-typecasting/. [Accessed: 11-Jan-2024].

[6] M. MKS075 Follow, “Difference between type casting and type conversion,” GeeksforGeeks, 20-Apr-2020. [Online]. Available: https://www.geeksforgeeks.org/difference-between-type-casting-and-type-conversion/. [Accessed: 11-Jan-2024].

[7] “C pointers,” GeeksforGeeks, 15-Dec-2016. [Online]. Available: https://www.geeksforgeeks.org/c-pointers/. [Accessed: 11-Jan-2024].

[8] “Decision making in C (if , if..Else, nested if, if-else-if),” GeeksforGeeks, 19-May-2017. [Online]. Available: https://www.geeksforgeeks.org/decision-making-c-cpp/. [Accessed: 11-Jan-2024].

[9] K. Follow, “C - loops,” GeeksforGeeks, 08-Oct-2022. [Online]. Available: https://www.geeksforgeeks.org/c-loops/. [Accessed: 11-Jan-2024].

[10] “C arrays,” GeeksforGeeks, 14-May-2015. [Online]. Available: https://www.geeksforgeeks.org/c-arrays/. [Accessed: 11-Jan-2024].

[11] “Strings in C,” GeeksforGeeks, 01-Aug-2017. [Online]. Available: https://www.geeksforgeeks.org/strings-in-c/. [Accessed: 11-Jan-2024].

[12] D. divya_dashrath_barvekar Follow, “Macros and its types in C,” GeeksforGeeks, 26-Jun-2020. [Online]. Available: https://www.geeksforgeeks.org/macros-and-its-types-in-c-cpp/. [Accessed: 11-Jan-2024].

[13] “C structures (structs),” W3schools.com. [Online]. Available: https://www.w3schools.com/c/c_structs.php. [Accessed: 11-Jan-2024].