What is a Data Type in C?

A data type in C refers to the type of data used to store the information. For example, the name of a person would be an array of characters, while the age will be in integers. Whereas, the marks of a student would require a data type that can store decimal values.

In C language, four different data types can be used to differentiate and store various types of data. They are given in the table below:

Type	Data types
Basic data types	int, char, float, & double
Derived data types	array, pointer, structure, & union
Enumeration data type	enum
Void data type	void

Below are the topics covered in this tutorial:

Basic Data Types

In C language, basic data types are used to store values in integer and decimal forms. It supports both signed and unsigned literals. There are four basic data types, in both signed and unsigned forms:

- Int
- Float
- Double
- Char

The memory size of these data types can change depending on the operating system (32-bit or 64-bit). Here is the table showing the data types commonly used in C programming with their storage size and value range, according to the 32-bit architecture.

Type	Storage Size	Value Range
Int (or signed int)	2 bytes	-32,768 to 32,767
unsigned int	2 bytes	0 to 65,535
Short int(or signed short int)	2 bytes	-32,768 to 32,767
Long(or singed short int)	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295
float	4 bytes	1.2E-38 to 3.4E+38 (6 decimal places)
double	8 bytes	2.3E-308 to 1.7E+308 (15 decimal places)
Long double	10 bytes	3.4E-4932 to 1.1E+4932 (19 decimal places)
char(or signed char)	1 byte	-128 to 127
unsigned char	1 byte	0 to 255

1. Integer Data Type

An integer type variable can store zero, positive, and negative values without any decimal. In C language, the integer data type is represented by the '**int**' keyword, and it can be both signed or unsigned. By default, the value assigned to an integer variable is considered positive if it is unsigned.

The integer data type is further divided into **short**, **int**, and **long** data types. The short data type takes 2 bytes of storage space; int takes 2 or 4 bytes, and long takes 8 bytes in 64-bit and 4 bytes in the 32-bit operating system.

If you try to assign a decimal value to the integer variable, the value after the decimal will be truncated, and only the whole number gets assigned to the variable. Here is an example that will help you understand more about the concept.

```
#include
int main() {
	short int a=10000;
	int b=11252486;
	long num1=499962313469;
	long long num2=51454456154585454;
```

```
      printf("a is %hd \n b is %d \n num1 is %ld \n num2 is
%lld \n",a,b,num1,num2);
      return 0;
}
```

The output of the above program will be as follows:

```
a is 10000
 b is 11252486
 num1 is 1746107133
 num2 is 51454456154585454
```

While doing an arithmetic operation, if the result comes out to be a decimal value, the variable will only accept the whole number and discard the numbers after the decimal point. For short int, an incorrect value will be displayed if the number is bigger than 10000.

2. Float-pointing Data Types

The floating-point data type allows a user to store decimal values in a variable. It is of two types:

- Float
- Double

Float

Float variables store decimal values with up to 6 digits after the decimal place. The storage size of the float variable is 4 bytes, but the size may vary for different processors, the same as the ‘int’ data type.

In C language, the **float** values are represented by the ‘**%f**’ format specifier. A variable containing integer value will also be printed in the floating type with redundant zeroes. It will be clear for you from the example given below:

```
#include
int main() {
      float sum=9664.35;
      float num=67;
      float average=(sum/num);
```

```
      printf("Average is %f \n", average);
      printf("Value of num is %f \n",num);
      printf("Value of num presented as an integer %d
\n",num);
}
```

The output of the above program will be as given below:

```
Average is 144.244019
Value of num is 67.000000
Value of num presented as an integer 0
```

If you assign an integer value to a float variable, the result will always be a float value with zeroes after the decimal place. As mentioned, float values are represented by the '%f' format specifier. However, if you try to print float values with '%d', then the output will not be 67 in the above case. Instead, you will see a garbage value on the output screen.

Double

The double data type is similar to float, but here, you can have up to 10 digits after the decimal place. It is represented by the 'double' keyword and is mainly used in scientific programs where extreme precision is required.

In the example below, the double variable prints the exact value '349999999.454', while the float variable messes up with the number and prints a round-off value.

```
#include
int main()
{
      float score=349999999.454;
      double average=349999999.454;
      printf("Score in %f \n",score);
      printf("Average is %lf",average);
      return 0;
}
```

The following will be the output of the above program:

```
Score in 350000000.000000
```

```
Average is 349999999.454000
```

3. Char

Char is used to storing single character values, including numerical values. If you create an array of the character data type, it becomes a string that can store values such as name, subject, and more.

Here is an example of creating an array of characters in C programming.

```
#include
int main()
{
      char group='A';
      // to store a string of characters in C programming, we use the array of the characters
      char name[30]="student";
      printf("The group is %c \n",group);
      printf("The name is %s",name);
      return 0;
}
```

The output of this program will be as given below:

```
The group is A
The name is student
```

The format specifier for a single character is ‘%c’, whereas, a string is represented by ‘%s’ format specifier.

Derived Data Types

The primitive or basic data types are used to store single values of different forms, but what if you need to store more values of the same data type? Here, derived data types allow you to combine the basic data types and store multiple values in a single variable.

Derived data types are defined by the user itself, which means that you can aggregate as many elements of similar data types as required. There are four types of derived data types:

- Array
- Pointer
- Structure
- Union

1. Array

An array is a collection of similar data-type elements stored in a contiguous memory location. For example, you can store float values like the marks of a student, integer values like the roll number, and so on.

Career Transition

-

See the below program to understand more:

```
#include
int main() {
	int ar[]={7,5,3,8,9};
	int i,j;
	float sum=0,average=0;
	printf("The array elements are: \n");
	for(i=0;i<5;i++){
		printf("%d \t",ar[i]);
	}
	int marks[5];
	printf("\nEnter Marks: \n");
	for(j=0;j<5;j++) {
		scanf("%d",&marks[i]);
		sum =sum + marks[i];
	}
	average=(sum/5);
	printf("The average is %f",average);
	return 0;
}
```

The output of the program will be like this:

```
The array elements are:
7       5       3       8       9
Enter Marks:
20
15
35
40
12
The average is 24.400000
```

In this example, the array either needs to be initialized or have the size while declaring it. The size of the array will depend on the data type you are using. Also, the default value of an integer array will be zero.

For accessing or changing specific elements in the program, marks[i] is used where ‘marks’ is the array name, in which ‘[i]’ denotes the position of the element.

2. Pointer

Pointer is a variable used to store the address of another variable. To store the address of a variable, the pointer variable should be of the same data type.

Pointer enables a user to perform dynamic memory allocation in C language and also pass variables by reference, which means that the user can pass the pointer having the address of the variable.

A pointer with no address is known as the null pointer, while a pointer with no data type is called a void or general-purpose pointer.

Have a look at the below program to understand this data type in-depth:

```
#include
int main() {
      int *ptr1;
      int *ptr2;
      int x=5,y=10;
      ptr1=&x; //Address of x assigned to ptr1
      ptr2=&y; //address of y assigned to ptr2
```

```
      printf(" Value of ptr1 %d", *ptr1);
      printf("\n Value of ptr2 %d", *ptr2);

      printf("\n Address of variable x %d",ptr1);
      printf("\n Address of variable y %d",ptr2);

      int add=(*ptr1 + *ptr2);
      printf("\n Addition of x and y using ptr1 and ptr2
%d",add);
      return 0;
}
```

The following will be the output of this program:

```
 Value of ptr1 5
 Value of ptr2 10
 Address of variable x 6487560
 Address of variable y 6487556
 Addition of x and y using ptr1 and ptr2 15
```

In C language, the difference operator '&' is used to assign the address of a variable to the pointer. To access the value stored in the address that the pointer is referring to, you should use the '*' operator.

3. Structure

In C language, a structure is a user-defined data type, which is a group of items used to store the values of similar or different data types. For example, structures can be used to store information about a student, including the name, the roll number, marks, and more. The record of each student will be represented by an object of the structure.

A structure can be created inside as well as outside the main method. To define a structure, the keyword 'struct' is used as shown in the below program. The size of the structure would be the sum of the storage size required by every variable.

```
#include
struct student{
```

```
	char *name;
	int roll_no;
	float avg_marks;
};

int main()
{
	struct student obj;
	obj.name="Steve";
	obj.roll_no=25;
	obj.avg_marks=85.32;

	printf("Student name is %s",obj.name);
	printf("\nStudent Roll No is %d",obj.roll_no);
	printf("\nStudent Average marks are %f",obj.avg_marks);

	return 0;
}
```

See the output of the above program:

```
Student name is Steve
Student Roll No is 25
Student Average marks are 85.320000
```

Courses you may

4. Union

Union is also a collection of elements with similar or different data types, but the memory location is the same for all the elements. It is a special kind of data type in C language, where you can declare many variables, but only one variable can store the value at a given time.

Union is defined by the ‘union’ keyword, where each object will represent a single record. The size of a union will be equal to the memory needed for the largest variable inside it. Let’s make things more clear with an example:

```
#include
union Data{
	char *name;
	int roll_no;
	float avg_marks;
};
int main() {
	union Data ob;
	ob.name="George";
	printf("Student name = %s",ob.name);
	ob.roll_no=5;
	printf("\nRoll No = %d",ob.roll_no);
	ob.avg_marks=82.5;
	printf("\nAverage Marks = %f",ob.avg_marks);

	printf("\nRoll Number = %d",ob.roll_no);
	return 0;
}
```

Have a look at the output of the program:

```
Student name = George
Roll No = 5
Average Marks = 82.500000
Roll Number = 1118109696
```

In the above example, the union student has three variables, name, roll no, and marks. The variable name can have a maximum of 25 characters, which means that the size would be 25 bytes. Similarly, roll no will have 4 bytes, and the average marks will also have 4 bytes.

Since the variable with maximum storage size here is ‘name’, the size of the union student will be 25 bytes. If the variable ‘name’ has some value in it and you assign a value to the roll no, the value in the ‘name’ will be replaced by the ‘roll no’ in the memory.

Enumeration Data Type

Enumeration is a user-defined data type used to assign names to the integral constants and enhance the readability of your program. The keyword used for enumeration is ‘enum’ with a syntax similar to structure:

```
enum flag{const1, const2, const3…….};
```

There are two main reasons why enumerations are better than ‘#define’:

- Enum constants get default values by the compiler
- They can be declared in the local scope

In the example below, the flag ‘week’ is now a data type with Monday, Tuesday, Wednesday, and so on as the integral constants. If you do not assign any value to the enum constant, then const1 will be 0, const2 will be 1, and so on. You can also define other variables with this enum data type as mentioned in the program:

```
#include
enum week{Monday, Tuesday, Wednesday, Thursday, Friday,
Saturday,Sunday};
int main(){
      printf("Value of Wednesday %d",Wednesday);
      enum week day;
      day=Saturday;
      printf("\n Value stored in the variable day %d",day);
      printf("\n The value of each enum constant: \n");
      for(int i=Monday;i<=Sunday;i++)
      {
            printf("%d \t",i);
      }
      return 0;
}
```

This will be the output of the program:

```
Value of Wednesday 2
```

```
 Value stored in the variable day 5
 The value of each enum constant:
0       1       2       3       4       5       6
```

Void Data Type

Void is a data type in C language that does not refer to any value of any type. It is mostly used as the return type in functions. You can declare the void pointers to take the address of variables from any data type. These pointers are often called ‘generic pointers.’

Look at the below example of the void data type:

```
#include
void addition(int a, int b);
int main(){
      int x=10;
      int y=20;
      addition(x,y);
      void *ptr;
      ptr=&x;
      printf("\n Value stored in pointer(ptr) after
dereferencing %d",*((int *)ptr));
      return 0;
}
void addition(int a,int b){
      int sum=a+b;
      printf("The sum of x and y is %d",sum);
}
```

See the output of the above program:

```
The sum of x and y is 30
 Value stored in pointer(ptr) after dereferencing 10
```

In this example, the function named ‘addition’ has void as the return type, which means that it will not return any value to the main method. So, the user has to print the message inside the function body or write a different return type.

In the main method, the void pointer ‘ptr’ acts as a generic pointer used to store the address of a variable of any data type. The output clearly shows that the ptr value changes from some garbage value to the value stored in the address it is pointing to (in this case, the value of the variable x).

Conclusion

In this tutorial, you learned about all the data types used in the C language and how they are further divided into subcategories. Also, the C programs given for each data type show how you can properly implement them and what would be the results if you change the properties of any data type.