**C struct**

In this tutorial, you'll learn about struct types in C Programming with the help of examples.

In C programming, a struct (or structure) is a collection of variables (can be of different types) under a single name.

**Define Structures**

Before you can create structure variables, you need to define its data type. To define a struct, the struct keyword is used.

**Syntax of struct**

struct structureName {

dataType member1;

dataType member2;

...

};

For example,

struct Person {

char name[50];

int citNo;

float salary;

};

Here, a derived type struct Person is defined. Now, you can create variables of this type.

**Create struct Variables**

When a struct type is declared, no storage or memory is allocated. To allocate memory of a given structure type and work with it, we need to create variables.

Here's how we create structure variables:

struct Person {

// code

};

int main() {

struct Person person1, person2, p[20];

return 0;

}

Another way of creating a struct variable is:

struct Person {

// code

} person1, person2, p[20];

In both cases,

* person1 and person2 are struct Person variables
* p[] is a struct Person array of size 20.

**Access Members of a Structure**

There are two types of operators used for accessing members of a structure.

1. . - Member operator
2. -> - Structure pointer operator (will be discussed in the next tutorial)

Suppose, you want to access the salary of person2. Here's how you can do it.

person2.salary

**Example 1: C++ structs**

#include <stdio.h>

#include <string.h>

// create struct with person1 variable

struct Person {

char name[50];

int citNo;

float salary;

} person1;

int main() {

// assign value to name of person1

strcpy(person1.name, "George Orwell");

// assign values to other person1 variables

person1.citNo = 1984;

person1. salary = 2500;

// print struct variables

printf("Name: %s\n", person1.name);

printf("Citizenship No.: %d\n", person1.citNo);

printf("Salary: %.2f", person1.salary);

return 0;

}

**Output**

Name: George Orwell

Citizenship No.: 1984

Salary: 2500.00

In this program, we have created a struct named Person. We have also created a variable of Person named person1.

In main(), we have assigned values to the variables defined in Person for the person1 object.

strcpy(person1.name, "George Orwell");

person1.citNo = 1984;

person1. salary = 2500;

Notice that we have used strcpy() function to assign the value to person1.name.

This is because name is a char array ([C-string](https://www.programiz.com/c-programming/c-strings)) and we cannot use the assignment operator = with it after we have declared the string.

**Keyword typedef**

We use the typedef keyword to create an alias name for data types. It is commonly used with structures to simplify the syntax of declaring variables.

For example, let us look at the following code:

struct Distance{

int feet;

float inch;

};

int main() {

struct Distance d1, d2;

}

We can use typedef to write an equivalent code with a simplified syntax:

typedef struct Distance {

int feet;

float inch;

} distances;

int main() {

distances d1, d2;

}

**Example 2: C++ typedef**

#include <stdio.h>

#include <string.h>

// struct with typedef person

typedef struct Person {

char name[50];

int citNo;

float salary;

} person;

int main() {

// create Person variable

person p1;

// assign value to name of p1

strcpy(p1.name, "George Orwell");

// assign values to other p1 variables

p1.citNo = 1984;

p1. salary = 2500;

// print struct variables

printf("Name: %s\n", p1.name);

printf("Citizenship No.: %d\n", p1.citNo);

printf("Salary: %.2f", p1.salary);

return 0;

}

**Output**

Name: George Orwell

Citizenship No.: 1984

Salary: 2500.00

Here, we have used typedef with the Person structure to create an alias person.

// struct with typedef person

typedef struct Person {

// code

} person;

Now, we can simply declare a Person variable using the person alias:

// equivalent to struct Person p1

person p1;

**Nested Structures**

You can create structures within a structure in C programming. For example,

struct complex {

int imag;

float real;

};

struct number {

struct complex comp;

int integers;

} num1, num2;

Suppose, you want to set imag of num2 variable to **11**. Here's how you can do it:

num2.comp.imag = 11;

**Example 3: C++ Nested Structures**

#include <stdio.h>

struct complex {

int imag;

float real;

};

struct number {

struct complex comp;

int integer;

} num1;

int main() {

// initialize complex variables

num1.comp.imag = 11;

num1.comp.real = 5.25;

// initialize number variable

num1.integer = 6;

// print struct variables

printf("Imaginary Part: %d\n", num1.comp.imag);

printf("Real Part: %.2f\n", num1.comp.real);

printf("Integer: %d", num1.integer);

return 0;

}

**Output**

Imaginary Part: 11

Real Part: 5.25

Integer: 6

**Why structs in C?**

Suppose, you want to store information about a person: his/her name, citizenship number, and salary. You can create different variables name, citNo and salary to store this information.

What if you need to store information of more than one person? Now, you need to create different variables for each information per person: name1, citNo1, salary1, name2, citNo2, salary2, etc.

A better approach would be to have a collection of all related information under a single name Person structure and use it for every person.

**C structs and Pointers**

In this tutorial, you'll learn to use pointers to access members of structs in C programming. You will also learn to dynamically allocate memory of struct types.

Before you learn about how pointers can be used with structs, be sure to check these tutorials:

* [C Pointers](https://www.programiz.com/c-programming/c-pointers)
* [C struct](https://www.programiz.com/c-programming/c-structures)

**C Pointers to struct**

Here's how you can create pointers to structs.

struct name {

member1;

member2;

.

.

};

int main()

{

struct name \*ptr, Harry;

}

Here, ptr is a pointer to struct.

**Example: Access members using Pointer**

To access members of a structure using pointers, we use the -> operator.

#include <stdio.h>

struct person

{

int age;

float weight;

};

int main()

{

struct person \*personPtr, person1;

personPtr = &person1;

printf("Enter age: ");

scanf("%d", &personPtr->age);

printf("Enter weight: ");

scanf("%f", &personPtr->weight);

printf("Displaying:\n");

printf("Age: %d\n", personPtr->age);

printf("weight: %f", personPtr->weight);

return 0;

}

In this example, the address of person1 is stored in the personPtr pointer using personPtr = &person1;.

Now, you can access the members of person1 using the personPtr pointer.

By the way,

* personPtr->age is equivalent to (\*personPtr).age
* personPtr->weight is equivalent to (\*personPtr).weight

**Dynamic memory allocation of structs**

Before you proceed this section, we recommend you to check [C dynamic memory allocation](https://www.programiz.com/c-programming/c-dynamic-memory-allocation).

Sometimes, the number of struct variables you declared may be insufficient. You may need to allocate memory during run-time. Here's how you can achieve this in C programming.

**Example: Dynamic memory allocation of structs**

#include <stdio.h>

#include <stdlib.h>

struct person {

int age;

float weight;

char name[30];

};

int main()

{

struct person \*ptr;

int i, n;

printf("Enter the number of persons: ");

scanf("%d", &n);

// allocating memory for n numbers of struct person

ptr = (struct person\*) malloc(n \* sizeof(struct person));

for(i = 0; i < n; ++i)

{

printf("Enter first name and age respectively: ");

// To access members of 1st struct person,

// ptr->name and ptr->age is used

// To access members of 2nd struct person,

// (ptr+1)->name and (ptr+1)->age is used

scanf("%s %d", (ptr+i)->name, &(ptr+i)->age);

}

printf("Displaying Information:\n");

for(i = 0; i < n; ++i)

printf("Name: %s\tAge: %d\n", (ptr+i)->name, (ptr+i)->age);

return 0;

}

When you run the program, the output will be:

Enter the number of persons: 2

Enter first name and age respectively: Harry 24

Enter first name and age respectively: Gary 32

Displaying Information:

Name: Harry Age: 24

Name: Gary Age: 32

In the above example, n number of struct variables are created where n is entered by the user.

To allocate the memory for n number of struct person, we used,

ptr = (struct person\*) malloc(n \* sizeof(struct person));

Then, we used the ptr pointer to access elements of person.

**C Structure and Function**

In this tutorial, you'll learn to pass struct variables as arguments to a function. You will learn to return struct from a function with the help of examples.

Similar to variables of built-in types, you can also pass structure variables to a function.

**Passing structs to functions**

We recommended you to learn these tutorials before you learn how to pass structs to functions.

* [C structures](https://www.programiz.com/c-programming/c-structures)
* [C functions](https://www.programiz.com/c-programming/c-functions)
* [User-defined Function](https://www.programiz.com/c-programming/c-user-defined-functions)

Here's how you can pass structures to a function

#include <stdio.h>

struct student {

char name[50];

int age;

};

// function prototype

void display(struct student s);

int main() {

struct student s1;

printf("Enter name: ");

// read string input from the user until \n is entered

// \n is discarded

scanf("%[^\n]%\*c", s1.name);

printf("Enter age: ");

scanf("%d", &s1.age);

display(s1); // passing struct as an argument

return 0;

}

void display(struct student s) {

printf("\nDisplaying information\n");

printf("Name: %s", s.name);

printf("\nAge: %d", s.age);

}

**Output**

Enter name: Bond

Enter age: 13

Displaying information

Name: Bond

Age: 13

Here, a struct variable s1 of type struct student is created. The variable is passed to the display() function using display(s1); statement.

**Return struct from a function**

Here's how you can return structure from a function:

#include <stdio.h>

struct student

{

char name[50];

int age;

};

// function prototype

struct student getInformation();

int main()

{

struct student s;

s = getInformation();

printf("\nDisplaying information\n");

printf("Name: %s", s.name);

printf("\nRoll: %d", s.age);

return 0;

}

struct student getInformation()

{

struct student s1;

printf("Enter name: ");

scanf ("%[^\n]%\*c", s1.name);

printf("Enter age: ");

scanf("%d", &s1.age);

return s1;

}

Here, the getInformation() function is called using s = getInformation(); statement. The function returns a structure of type struct student. The returned structure is displayed from the main() function.

Notice that, the return type of getInformation() is also struct student.

## Passing struct by reference

You can also pass structs by reference (in a similar way like you pass variables of built-in type by reference). We suggest you to read [pass by reference](https://www.programiz.com/c-programming/c-pointer-functions) tutorial before you proceed.

During pass by reference, the memory addresses of struct variables are passed to the function.

#include <stdio.h>

typedef struct Complex

{

float real;

float imag;

} complex;

void addNumbers(complex c1, complex c2, complex \*result);

int main()

{

complex c1, c2, result;

printf("For first number,\n");

printf("Enter real part: ");

scanf("%f", &c1.real);

printf("Enter imaginary part: ");

scanf("%f", &c1.imag);

printf("For second number, \n");

printf("Enter real part: ");

scanf("%f", &c2.real);

printf("Enter imaginary part: ");

scanf("%f", &c2.imag);

addNumbers(c1, c2, &result);

printf("\nresult.real = %.1f\n", result.real);

printf("result.imag = %.1f", result.imag);

return 0;

}

void addNumbers(complex c1, complex c2, complex \*result)

{

result->real = c1.real + c2.real;

result->imag = c1.imag + c2.imag;

}

**Output**

For first number,

Enter real part: 1.1

Enter imaginary part: -2.4

For second number,

Enter real part: 3.4

Enter imaginary part: -3.2

result.real = 4.5

result.imag = -5.6

In the above program, three structure variables c1, c2 and the address of result is passed to the addNumbers() function. Here, result is passed by reference.

When the result variable inside the addNumbers() is altered, the result variable inside the main() function is also altered accordingly.

**C Unions**

In this tutorial, you'll learn about unions in C programming. More specifically, how to create unions, access its members and learn the differences between unions and structures.

A union is a user-defined type similar to [structs in C](https://www.programiz.com/c-programming/c-structures) except for one key difference.

Structures allocate enough space to store all their members, whereas **unions can only hold one member value at a time**.

**How to define a union?**

We use the union keyword to define unions. Here's an example:

union car

{

char name[50];

int price;

};

The above code defines a derived type union car.

**Create union variables**

When a union is defined, it creates a user-defined type. However, no memory is allocated. To allocate memory for a given union type and work with it, we need to create variables.

Here's how we create union variables.

union car

{

char name[50];

int price;

};

int main()

{

union car car1, car2, \*car3;

return 0;

}

Another way of creating union variables is:

union car

{

char name[50];

int price;

} car1, car2, \*car3;

In both cases, union variables car1, car2, and a union pointer car3 of union car type are created.

**Access members of a union**

We use the . operator to access members of a union. And to access pointer variables, we use the -> operator.

In the above example,

* To access price for car1, car1.price is used.
* To access price using car3, either (\*car3).price or car3->price can be used.

**Difference between unions and structures**

Let's take an example to demonstrate the difference between unions and structures:

#include <stdio.h>

union unionJob

{

//defining a union

char name[32];

float salary;

int workerNo;

} uJob;

struct structJob

{

char name[32];

float salary;

int workerNo;

} sJob;

int main()

{

printf("size of union = %d bytes", sizeof(uJob));

printf("\nsize of structure = %d bytes", sizeof(sJob));

return 0;

}

**Output**

size of union = 32

size of structure = 40

**Why this difference in the size of union and structure variables?**

Here, the size of sJob is 40 bytes because

* the size of name[32] is 32 bytes
* the size of salary is 4 bytes
* the size of workerNo is 4 bytes

However, the size of uJob is 32 bytes. It's because the size of a union variable will always be the size of its largest element. In the above example, the size of its largest element, (name[32]), is 32 bytes.

With a union, all members share **the same memory**.

**Example: Accessing Union Members**

#include <stdio.h>

union Job {

float salary;

int workerNo;

} j;

int main() {

j.salary = 12.3;

// when j.workerNo is assigned a value,

// j.salary will no longer hold 12.3

j.workerNo = 100;

printf("Salary = %.1f\n", j.salary);

printf("Number of workers = %d", j.workerNo);

return 0;

}

**Output**

Salary = 0.0

Number of workers = 100