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
- C struct

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.

```c
#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;
}
```

Run Code

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.

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

```c
#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;
}
```

Run Code

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
- C functions
- User-defined Function

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);
}
```

Run Code

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;
}
```

Run Code

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 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;
}
```
Run Code

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.