1 Typedef

In C, ***typedef*** allows us to define an alias to another type. This is how **size\_t**is defined. For example, we can define a type **mySize\_t** as follows.

**typedef** unsigned int mySize\_t;

Then, we can use **mySize\_t**as we would any type. For example,

mySize\_t x = 8;  
printf("%d\n", x);

Copy and paste the following code into "typedef.c", compile and run it.

#include <stdio.h>   
  
**typedef** unsigned int **mySize\_t**;   
  
int main()   
{   
 **mySize\_t** x = 8;   
 printf("%d\n", x);   
  
 return 0;   
}

Typedefs are useful for defining aliases to types so that the type can be easily changed.

2 Struct

In C, ***structs***are a way of organizing data. They are similar to classes in Java, only simpler since they are only used to group variables.

The following code defines a **Date**struct.

**struct** Date {  
 **int** month;  
 **int** day;  
 **int** year;  
};

To declare a struct variable, use:

**struct** Date halloween;

Declaring a struct will allocate memory for the struct.

To declare and initialize a struct, use:

**struct** Date halloween = {10, 31, 2020};

Access the fields of a struct as follows

halloween.month = 10; // assign a field  
printf("%d\n", halloween.month); // access a field

Paste the following code in ***struct1.c*** Review, compile, and run the code.

#include <stdio.h>  
  
struct Date  
{  
    int month;  
    int day;  
    int year;  
};  
  
int main() {  
  
    struct Date halloween = {11, 31, 2021};  
  
    printf("%d\n", halloween.month);  
    printf("%d\n", halloween.day);  
  
    return 0;  
}

2.1 Structs and typedef

The syntax

**struct** Date halloween = {10, 31, 2020};

is inconvenient.

Typedef can be used to make declaring structs easier.

Paste the following code in ***struct2.c*** Review, compile, and run the code.

#include <stdio.h>  
  
**typedef** struct Date  
{  
 int month;  
 int day;  
 int year;  
} **Date**;  
  
int main() {  
  
 **Date** halloween = {11, 31, 2021};  
  
 printf("%d\n", halloween.month);  
 printf("%d\n", halloween.day);  
  
 return 0;  
}

Note that the Date struct is declared using "Date" instead of "struct Date".

2.2 Structs and pointers

When you have a pointer to a struct, there are two ways to access the data in the struct. Let's look at them.

Paste the following code into ***struct3.c***Review, compile, and run the code.

#include <stdio.h>  
  
typedef struct Date  
{  
 int month;  
 int day;  
 int year;  
} Date;  
  
int main()  
{  
 Date halloween = {11, 31, 2021};  
 **Date \*ptr = &halloween;**  
  
 printf("%d\n", **(\*ptr).month**);  
 printf("%d\n", **ptr->month**);  
  
 return 0;  
}

The first way to access the data in the struct using a pointer to the struct is to dereference the struct pointer and using dot, '.', operator like this

**(\*ptr).month**

The other way is to using the arrow ("->") operator like this

**ptr->day**

2.3 Pointers in structs

Struct can encapsulate any datatype including pointers. Let's look at an example.

Paste the following code into ***struct4.c***Review, compile, and run the code.

#include <stdio.h>  
  
typedef struct  
{  
 **int \*ptrX;**  
 **int \*ptrY;**  
} Pair;  
  
int main()  
{  
 int x = 10, y = 20;  
 Pair p = {**&x**, **&y**};  
  
 printf("(%d, %d)\n", **\*(p.ptrX)**, **\*(p.ptrY)** );  
  
 return 0;  
}