C

Structures and Functions

Patrick Stiller

November 3, 2014

Introduction

A function is a group of statements. Every function has a return value. Every *C* Program has at least one function , which is **main()**.

Defining functions

```
1  return_type function_name(parameter list)
2  {
3      body of the function
4
5      return // Same type like return_type
6  }
```

Explanation

- The Return Type is the data type of the value the function returns
- The function Name is the actual name of the function
- Parameters are placeholder. If you call the function,you want to give the values of the Parameters. You can work with parameters like variables but the scope of the parameters is only in the function.
- The function body defines what the function does.

Example

```
1  // little Function multiplying x and y
2  int mult(int x,int y){
3    return x * y ;
4  }
```

Call of functions

The function name and the parameter list together constitute the function signature.

```
1  #include <stdio.h>
2
3  int mult(int x, int y){
4        return x * y ;
5  }
6
7  int main(int argc, char *argv[]){
8    int zahl1 = 12;
9    int zahl2 = 13;
10   int erg;
11   erg = mult(zahl1,zahl2); // has the value 156
12   return 0 ;
13 }
```

Prototype

The scope of a function is from the beginning of the function till the end of the program. If you want a bigger scope you need a prototype of the function.

Definition

```
1 ReturnType functionName(parameterType param1,...,
      parameterType param2);
```

An Example

```
1  #include <stdio.h>
2
3  int mult ( int x, int y );
4
5  int main(int argc, char *argv[]){
6      int x = 122;
7      int y = 33;
8      int erg ;
9      erg = mult (x,y);
10     return 0 ;
11 }
12
13 int mult (int x, int y)
14 {
15     return x * y;
16 }
```

Introduction

Sometimes you need a datatype which store many different values in variables of potentially different types under the same name. With structures is that possible.

Definitions

The format for defining a structure is

```
1 struct Tag {
2     type1 member1;
3     type2 member2;
4     type3 member3;
5     ...
6 };
```

Struct members

A member is a variable with any datatype. A struct could be a member too(datastructures).

Creating of structures and access to the members

Creating of structures

```
1 /*
2 create a single structure
3 */
4 struct Tag nameOfSingleStructure;
```

Access to the members

```
1 /*
2 Access to the members
3 */
4 nameOfSingleStructure.NameOfVariable;
```

An example : point

A little program printing the values of x and y.

```
1  struct Point{
2      int x;
3      int y;
4  };
5
6  int main(int argc, char *argv[]){
7      struct Point point; // Create a Point
8      point.x = 2; // Set the value of x
9      point.y = 3; // Set the value of y
10     //alternative : struct Point point ={2,3};
11     printf("Point x: %d  y: %d",point.x ,point.y);
12     return 0;
13 }
```

An example with different datatypes

```
1  #include <stdio.h>
2
3  struct Person{
4      int id;
5      int age;
6      double weight;
7  };
8
9  int main(int argc, char *argv[]){
10     struct Person person1;
11     person1.age = 20;
12     person1.id = 12;
13     person1.weight = 72.56;
14     return 0;
15 }
```