**Data structures.**

A *data structure* is a group of data elements grouped together under one name. These data elements, known as *members*, can have different types and different lengths. Data structures can be declared in C++ using the following syntax:

struct type\_name {  
member\_type1 member\_name1;  
member\_type2 member\_name2;  
member\_type3 member\_name3;  
.  
.  
} object\_names;

Where *type\_name* is a name for the structure type, *object\_name* can be a set of valid identifiers for objects that have the type of this structure. Within braces {}, there is a list with the data members, each one is specified with a type and a valid identifier as its name.

For example:

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 5 6 7 | struct product {  int weight;  double price;  } ;  product apple;  product banana, melon; |  |

This declares a structure type, called *product*, and defines it having two members: *weight* and *price*, each of a different fundamental type. This declaration creates a new type (*product*), which is then used to declare three objects (variables) of this type: *apple*, *banana*, and *melon*. Note how once *product* is declared, it is used just like any other type.

Right at the end of the *struct* definition, and before the ending semicolon (;), the optional field *object\_names* can be used to directly declare objects of the structure type. For example, the structure objects *apple*, *banana*, and *melon* can be declared at the moment the data structure type is defined:

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 | struct product {  int weight;  double price;  } apple, banana, melon; |  |

In this case, where *object\_names* are specified, the type name (*product*) becomes optional: *struct* requires either a *type\_name* or at least one name in *object\_names*, but not necessarily both.

It is important to clearly differentiate between what is the structure type name (*product*), and what is an object of this type (*apple*, *banana*, and *melon*). Many objects (such as *apple*, *banana*, and *melon*) can be declared from a single structure type (*product*).

Once the three objects of a determined structure type are declared (*apple*, *banana*, and *melon*) its members can be accessed directly. The syntax for that is simply to insert a dot (.) between the object name and the member name.

For example, we could operate with any of these elements as if they were standard variables of their respective types:

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 5 6 | apple.weight  apple.price  banana.weight  banana.price  melon.weight  melon.price |  |

Each one of these has the data type corresponding to the member they refer to: *apple.weight*, *banana.weight*, and *melon.weight* are of type *int*, while *apple.price*, *banana.price*, and *melon.price* are of type *double*.

Here is a real example with structure types in action:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 | // example about structures  #include <iostream>  #include <string>  #include <sstream>  using namespace std;  struct movies\_t {  string title;  int year;  } mine, yours;  void printmovie (movies\_t movie);  int main ()  {  string mystr;  mine.title = "2001 A Space Odyssey";  mine.year = 1968;  cout << "Enter title: ";  getline (cin,yours.title);  cout << "Enter year: ";  getline (cin,mystr);  stringstream(mystr) >> yours.year;  cout << "My favorite movie is:\n ";  printmovie (mine);  cout << "And yours is:\n ";  printmovie (yours);  return 0;  }  void printmovie (movies\_t movie)  {  cout << movie.title;  cout << " (" << movie.year << ")\n";  } | Enter title: Alien  Enter year: 1979  My favorite movie is:  2001 A Space Odyssey (1968)  And yours is:  Alien (1979) | [Edit & Run](https://www32.cplusplus.com/doc/tutorial/structures/) |

The example shows how the members of an object act just as regular variables. For example, the member *yours.year* is a valid variable of type *int*, and *mine.title* is a valid variable of type *string*.

But the objects *mine* and *yours* are also variables with a type (of type *movies*\_t). For example, both have been passed to function *printmovie* just as if they were simple variables. Therefore, one of the features of data structures is the ability to refer to both their members individually or to the entire structure as a whole. In both cases using the same identifier: the name of the structure.

Because structures are types, they can also be used as the type of arrays to construct tables or databases of them:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 | // array of structures  #include <iostream>  #include <string>  #include <sstream>  using namespace std;  struct movies\_t {  string title;  int year;  } films [3];  void printmovie (movies\_t movie);  int main ()  {  string mystr;  int n;  for (n=0; n<3; n++)  {  cout << "Enter title: ";  getline (cin,films[n].title);  cout << "Enter year: ";  getline (cin,mystr);  stringstream(mystr) >> films[n].year;  }  cout << "\nYou have entered these movies:\n";  for (n=0; n<3; n++)  printmovie (films[n]);  return 0;  }  void printmovie (movies\_t movie)  {  cout << movie.title;  cout << " (" << movie.year << ")\n";  } | Enter title: Blade Runner  Enter year: 1982  Enter title: The Matrix  Enter year: 1999  Enter title: Taxi Driver  Enter year: 1976    You have entered these movies:  Blade Runner (1982)  The Matrix (1999)  Taxi Driver (1976) | [Edit & Run](https://www32.cplusplus.com/doc/tutorial/structures/) |

The most efficient method of dealing with structure variables is to define the structure globally. This tells "the whole world", namely main and any functions in the program, that a new data type exists. However, the *struct* type can then be defined locally in main.