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.

```
struct type_name {
member_type1 member_name1;
member_type2 member_name2;
member_type3 member_name3;
.
.
} object_names;
```

```
1 struct product {
2   int weight;
3   double price;
4 } ;
5
6 product apple;
7 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.

The structure objects **apple, banana, melon** can be declared at the moment the data structure type is defined:

```
struct product {
  int weight;
  double price;
} apple, banana, melon;
```

Struct requires either a **type_name** or at least one name in **object_names**, but not necessarily both.

If you want to access directly the members, just put a (.) between **the object name** and **the member name.**

```
// 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)
```

The objects **mine** and **yours** are also **variables with a type (of type movies_t)**. Both have been passed to function **printmovie** just as if they were simple variables. Because **structures are types**, they can also be used as **the type of arrays** to construct tables or databases of them:

```
// 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)
```

Pointers To Structures (Like any other type, structures can be pointed to by its own type of pointers.)

```
struct movies_t {
  string title;
  int year;
};

movies_t amovie;
movies_t * pmovie;
```

Here **amovie** is an object of structure type movies_t, and **pmovie** is a pointer to point to objects of structure type movies_t.

Therefore, the following code would also be valid:
pmovie = &amovie //The value of the pointer **pmovie** would be assigned the address of object **amovie.**

```cpp
// pointers to structures
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

struct movies_t {
  string title;
  int year;
};

int main ()
{
  string mystr;

  movies_t amovie;
  movies_t * pmovie;
  pmovie = &amovie;

  cout << "Enter title: ";
  getline (cin, pmovie->title);
  cout << "Enter year: ";
  getline (cin, mystr);
  (stringstream) mystr >> pmovie->year;

  cout << "\nYou have entered:\n";
  cout << pmovie->title;
  cout << " (" << pmovie->year << ")\n";

  return 0;
}
```

```
Enter title: Invasion of the body snatchers
Enter year: 1978

You have entered:
Invasion of the body snatchers (1978)
```

The arrow operator **(->)** is a **dereference operator** that is used exclusively with pointers to objects that have members.
This operator serves to **access the member of an object directly from its address.**
pmovie **->** title //It is equivalent to **(*pmovie).title**

BUT IT'S DEFINITELY SOMETHING DIFFERENT THAN. ***pmovie.title** which is rather equivalent to ***(pmovie.title).**
This would access the value pointed by a hypothetical(varsayımsal) pointer member called **title** of the structure object **pmovie** (which is not the case, since **title** is not a pointer type).

The following panel summarizes possible combinations of the operators for pointers and for structure members:

Expression	What is evaluated	Equivalent
`a.b`	Member `b` of object `a`	
`a->b`	Member `b` of object pointed to by `a`	`(*a).b`
`*a.b`	Value pointed to by member `b` of object `a`	`*(a.b)`

Nesting Structures (In such a way that an element of a structure is itself another structure)

```
struct movies_t {
  string title;
  int year;
};

struct friends_t {
  string name;
  string email;
  movies_t favorite_movie;
} charlie, maria;

friends_t * pfriends = &charlie;
```

After the previous declarations, all of the following expressions would be valid:

```
charlie.name
maria.favorite_movie.title
charlie.favorite_movie.year
pfriends->favorite_movie.year
```

(where, by the way, the last two expressions refer to the same member).