STRUCTURES IN C/C++

struct → The most **flexible** way to represent data in C language (equivalent to a record in Pascal, or tuple),

→ a **user-defined**, complex data type declaration that defines a physically **grouped list** of variables

→ a group of variables (could combine **different types**), placed under one name, in a one **contiguous block** of memory, accessed via a **single name** or single pointer,

Definition of a new structural type:

```
struct  name_of_new_type                    ← name of the created stucture
        {
          type_name_1    field_name_1;
          type_name_2    field_name_2;      ← types and names of fields
             • • •
          type_name_n    field_name_n;
        } ;
```

Example:

Kowalski	Jan	1987	173 cm	'm'	437.20zł
family_name	name	birth_year	height	gender	scholarship
char [30]	**char [15]**	**short**	**short**	**char**	**double**

```
struct TPersonalData
  {
    char      family_name [30];
    char      name [15];
    short     birth_year, height;
    char      gender;
    double    scholarship;
  } ;

struct TCalendarDate   {   int day, month, year;   } ;

struct TLoan
  {
    double                amount;
    char                  description [50];
    TCalendarDate         loan_date;          // nested stucture
    TCalendarDate         payback_date;       // nested structure
  } ;
```

Reserving / defining variables:

```
struct  name_of_struct_type   name_of_new_variable;   // C
        name_of_struct_type   name_of_new_variable;   // C++
```

Examples:

```
TPersonalData   student_1;
TPersonalData   student_2,  student_3,  new_student;
TPersonalData   student_group [ 20 ];

TLoan   loan_for_Thomas;
TLoan   my_loans [ 100 ];
```

We can connect the variable definition and its initialization, Eg.:

```
                    // { family_name, name, birth_year, height, gender, scholarship }
TPersonalData   student_x = {"Kowalski", "Jan", 1970, 175 ,'M', 320.00 };

                    // { amount, description , { day, month, year }, { day, month, year } }
TLoan loan_x = { 100.00, "fee per apartment", {27,11,2006}, {3,12,2006}};
```

We can combine definition of **new type** and definition of **new variable**:

```
struct  new_type_name                ← this type name could be ommited
   {
     type_name _1     field_name _1;
     type_name _2     filed_name _2;          ← definitions of fields
       • • •
     type_name _n     field_name _n;
   } new_ variable ;                          ← name of a new variable
```

Eg.

```
struct                                        // ← skipped type name
  {
    char   family_name [30];
    char   name [15];
    short  birth_year, height;
    char   gender;
    double scholarship;
  } student_1, student_2;          // definition of two new variables / instances
```

Accessing the structure memebers: with a dot opearator

Examples:

```
student_1.height = 180 ;                    // assignment of numerical data
student_2.height = student_1.height;

loan_for_Thomas.amount = 50.00;
loan_for_Thomas.payback_date.month = 12;

scanf( "%lf" , &student.scholarship );      // input of a numerical data
scanf( "%s" , student.family_name );        // input a C-string  char[ ]

cin >> student.scholarship;
cin >> student.family_name;

strcpy( student.name, "Thomas" );   // copying the C-string into a structure field
```

In the original version of the C language (Kernighan, Ritchie),
the only operations permitted on the structure were: "the address of operator" (&)
and accesing the components/fields/memebers with a dot operator.

In the C ++, it is also possible to directly assign (copy) all structure content,
to pass the structure as the parameter to function, or return struct as a result.

```
memcpy( &student_2, &student_1, sizeof(student_1) );  // the only way in C !
student_2 = student_1;          // direct assignment of struct, introduced in C++
student_2.family_name = student_1.family_name;        // attention!  char [ ]
strcpy( student_2.family_name, student_1.family_name );
```

```
                                          // function which returns a struct
TPersonalData   Read_New_Personal_Data ( void )
  {
    TPersonalData  new_data;
    printf( "Enter family name: " );
    scanf( "%s" , new_data.family_name );
    • • •
    return new_data ;
  }

                                  // function which receives the struct as an argument
void Display_Personal_Data ( const TPersonalData  &person )
  {         // or alternatively a copy:    ( TPersonalData  person )
    printf( "Family name: %s\n" , person.family_name );
    printf( "          Name: %s\n" , person.name );
    • • •
  }
```

Struct as an element of Array:

```
TPersonalData   empty_struct_array [ 100 ];      // definitions of struct arrays

TPersonalData  database[ 3 ] = {
                                { "Kowalski", "Jan", 1970, 175, 'm', 95.00 } ,
                                { "Nowak", "Tomasz", 1965, 180, 'm', 0.00 },
                                { "Nowak", "Anna", 1983, 162, 'k', 250.0 } };

database [0].height = 175;              // example operations of database fields
database [1].family_name[0] = 'N';
strcpy( database[2].name, "Anna" );
```

	family_name	name	birth_date	height	gender	scholarship
0	Kowalski	Jan	1970	**175**	'm'	95.00
1	Nowak	Tomasz	1965	180	'm'	0.00
2	Nowak	Anna	1983	162	'k'	250.00

```
for( int i = 0; i < 3; i++ )                // display the all content of a "database"
  {
     printf( "Person no [%d] \n" , i+1 );
     printf( "Family name: %s \n" , database [ i ].family_name );
     • • •
     printf( "Scholarship: %.2f \n" , database [ i ].scholarship );
  }
```

Pointers to struct: an 'arrow' operator **->** to access the struct members

```
TPersonalData  student;
student.height = 180;                   // direct assignment to stuct member field

TPersonalData  *ptr_person;                                  // a pointer to struct
ptr_person = &student;

(*ptr_person).height = 180;    // traditional notation using  *  'star' operator
ptr_person->height = 180;                  // the same with an 'arrow' operator

cin >> database[ 2 ].scholarship;      //scanf("%lf",&(database[2].scholarship) );
cin >> (database+2)->scholarship;  // scanf("%lf",&((database+2)->scholarship));

for(ptr_person =database; ptr_person <database+3; ptr_person ++)
   cin >> ptr_person ->scholarship;
```

Example 1 : loading / viewing / filtering the list of loans

```
#include <iostream>                                   // implementation in C++
struct TCalendarDate { int day, month, year; };

struct TLoan { double amount;   char  description[50];
               TCalendarDate  loan_date, payback_date;};
const int N=10;
TLoan  my_loans [N];

int main()
{
   cout<<"Please enter the data of your loans:";
   for(int i=0; i<N; i++) {
      cout<<"\n\nLoan number "<< i <<endl;
      cout<<"Amount in PLN = ";      cin>>my_loans[i].amount;
      cout<<"Description  = ";       cin>>my_loans [i].description;
      cout<<"Date of loan:\n";
      cout<<"     day = ";           cin>>my_loans[i].loan_date.day;
      cout<<"  month = ";            cin>>my_loans[i].loan_date.month;
      cout<<"    year = ";           cin>>my_loans[i].loan_date.year;
   }
   cout<<"\n\nDisplaying the full list of entered loans:";
   for(int i=0; i<N; i++) {
      cout<<"\n\nLoan number "<< i <<endl;
      cout<<"Amount in PLN = "<< my_loans[i].amount ;
      cout<<" Description = "<< my_loans [i].description;
      cout<<"\nDate of loan: "<< my_loans[i].loan_date.day <<"/";
      cout<< my_loans[i].loan_date.month<<"/";
      cout<< my_loans[i].loan_date.year;
   }
   cout<<"\n\nDisplaying only the big loans (greater than 100PLN): ";
   for(int i=0; i<N; i++)
      if(my_loans[i].amount >100 ) {
         cout<<"\n\nLoan number "<< i <<endl;
         cout<<"Amount in PLN = "<< my_loans[i].amount ;
         cout<<" Description = "<< my_loans [i].description;
         cout<<"\nDate of loan: "<< my_loans[i].loan_date.day <<"/";
         cout<< my_loans[i].loan_date.month<<"/";
         cout<< my_loans[i].loan_date.year;
      }
   cout<<"\n\nEnd of the program. Press ENTER to close the console";
   cin.ignore(); cin.get();
   return 0;
}
```

Example 2 : Another version of Example no 1 → using the functions

```
#include <iostream>                                   // implementation in C++

// --- definition of calendar date struct and functions READ / DISPLAY ---

struct TCalendarDate { int day, month, year; };

void READ_DATE(TCalendarDate &  date)
{
   cout<<" day   = ";        cin>>date.day;
   cout<<" month = ";        cin>>date.month;
   cout<<" year  = ";        cin>>date.year;
}

void DISPLAY_DATE(const TCalendarDate &  date)
{
   cout<<date.day<<"/"<<date.month<<"/"<<date.year;
}
```

```
// ------ definition of loan struct and functions READ / DISPLAY for loans ------

struct TLoan { double amount; char description [50];
                        TCalendarDate loan_date, payback_date;};

void READ_LOAN (TLoan &  loan)
{
   cout<<"Amount      = ";  cin>>loan.amount;
   cout<<"Description = ";  cin>>loan.description;
   cout<<"Loan date:\n";
   READ_DATE( loan.loan_date );
   cout<<"Date of payback:\n";
   READ_DATE( loan.payback_date );
}

void DISPLAY_LOAN(const TLoan &  loan)
{
   cout << "Amount = " << loan.amount;
   cout << " Description = "  << loan.description;
   cout << "\nLoan date: ";
   DISPLAY_DATE( loan.loan_date );
   cout << "\nDate of payback: ";
   DISPLAY_DATE ( loan.payback_date );
}
```

. . . continuation of example (2)

```
// ------------------------------ Main program ---------------------------------
// ---------------- Using just defined structures and functions ----------------

// TCalendarDate  / READ_DATE  / DISPLAY_DATE
// TLoan  / READ_LOAN  / DISPLAY_LOAN


const inr N=10;
TLoan  my_loans[N];

int main()
{
   cout<<" Please enter the data of your loans:";
   for(int i=0; i<N; i++)
      {
         cout<<"\n\nLoan number "<< i <<endl;
         READ_LOAN( my_loans [i] );
      }

   cout<<"\n\n Displaying the full list of entered loans: ";
   for(int i=0; i<N; i++)
      {
         cout<<"\n\nLoan number "<<i<<endl;
         DISPLAY_LOAN( my_loans [i] );
      }

   cout<<"\n\nDisplaying only the big loans (greater than 100PLN):";
   for(int i=0; i<N; i++)
      if( my_loans[i].amount>100 )
         {
            cout<<"\n\nLoan number "<< i <<endl;
            DISPLAY_LOAN( my_loans[i] );
         }

   cout<<"\n\n nEnd of the program. Press ENTER to close the console";
   cin.ignore(); cin.get();
   return 0;
}
```