Algorithmic and Data Structure 1

Chapter 6 "Custom types"

Outline

Chapter 6: Custom Types

Why custom types?

In some problems, we can find data that is strongly related to each other, so treating one should directly affect the others. It is therefore appropriate to group this data into an indivisible set of information.

Example: we want to record information about customers to reuse them later. For each customer, the information are name, first name and age. Instead of recording each piece of information independently, we can group it into a single structure as follows:

Data structure
Customer Type Name: string First name: string Age: integer **EndType**

In this course, we are primarily interested in the two custom types: enumeration and records.

1. Enumeration (Enumerated type)

Enumerations allow you to define a type by the list of values it can take.

The enumerated type makes it possible to represent objects, which can take their value in a finite and ordered list, in other words, the enumerated type makes it possible to associate with a type a set of values ordered according to their order of declaration.

1.1. Declaration of enumerated type

To declare an enumerated type, we use a name for object identification (identifier), followed by a set of values in parentheses, as follows:

Syntax:

Type

Identifier = (Val_1, Val_2, Val_3, ….., Val_N);

- *Identifier* : the name of the enumeration;

- *Val_1, Val_2, Val_3, ….., Val_N*, are values that *Identifier can take* .
- *Val_1, Val_2, Val_3, ….., Val_N*, are ordered, i.e. $Val_1 < Val_2 < Val_3 < \ldots < Val_N$

Syntax in C:

```
enum identifier {Val 1 , Val 2 , Val 3 , ….., Val N };
```

Examples

- **Type** Week = (Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday);
- **Type** Color = (red, green, blue);
- **Type** Gender = (male, female);
- **Type** Vowel= (A, E, I, O, U);

1.2. Using an Enumerated Type

We use the enumerated type to declare variables of this type, this variable can only take one of the values given in parentheses.

Syntax

Variable Name _ Var : *Name _ Type _ Enumerate;*

Syntax in C

```
enum Name_Type_Enumere   Name_Var;
```

Example

- **variable** W: Week;
- **Variable** C: Color;
- **Variable** S: Gender;
- **Variable** V: Vowel;

Example in C

```
enum week {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday,
Saturday};
main()
{
enum week day;
day = Wednesday;
printf("Day %d",day+1);
}
```

***The program will display “** *Day 4”*

2. Records (Structures)

We saw in the previous chapter that the array type allows us to represent (save) several values of the same type. Whereas, the record (structure) is a data structure making it possible to group together in a single entity a set of data of different types associated with the same and single object.

The record is identified by a name and a set of properties called **fields.** Each field is identified by a name, which allows direct access to it and a type. The type can be simple or structured.

2.1. Declaration of a records

To declare a record we can use the following syntax:

Type *Record_Name =* ***Record***
Field_name1: Type1
Field_name2: Type2
…
Field_nameN: $Type_N$
EndRecord

NB: If there are fields of the same type, they can be declared together by separating them with commas.

To declare a record type variable, we can use the following declaration:

Variable Variable_name: Record_name;

Syntax in C

```
typedef struct Record_name
{Type1 Field1;
 Type2 Field2;
   …….
 Type N Field N ;
} Record_Name;
```

Example

A student's information: last name, first name, age, gender, result_BAC can be grouped into a record

```
Type Student= Record
  Name: string;
  First name: string;
  Age: integer
  Gender: character // 'M': Male, 'F': Female
  Result_Bac: real
EndRecord
```

```
Variables   student1, student2:  Student // Two Student type variables
            e1, e2, e3: Student // Three Student type variables
```

2.1.Accessing fields in a record

You can manipulate the fields of a record field by field. Access to the fields of a record is done by specifying the name of the record type variable followed by the name of the field separated by a point (. **)** :

Variable_name . Field_Name ;

Example : to modify the age of student1 using the assignment we must write:

student1.Age ← 22;

The point indicates the access path: We first access the student1 variable then we select the Age field.

2.2.Nested structures

A record can be nested in an array or record type structure, as it can have fields of any structured type (e.g. array). The notation used to select fields remains the same, using the point.

→ Record arrays

It is possible to declare an array whose elements are of record type. In this case, we first define the record type, then we declare an array whose elements are of this record type. As following:

```
Type Record_Name = Record
Field_name1: Type1
Field_name2: Type2
…
Field_nameN: TypeN
EndRecord
Variable array_name: array [1..N] record_name;
```

To access a box in the array, we use the brackets [ ], then we access the field using the point.

Example : to select the third field of the second element of the array we use the syntax:

Array_name[2] . Field_name3

Example

We want to declare an array of records to manipulate the information of 50 students.

```
Type Student = Record
Name: string;
First name: string;
Age: integer;
Gender: character
Result_Bac: real
EndRecord
Variable Tab: array [1..50] Student;
```

To modify the fields of student number 10, you can write:

Tab[10] . Name ← "Xxxxx"

Tab[10] . First name ← "Yyyyy"

Tab[10] . Age ← 17

Tab[10] . Gender ← 'M'

Tab[10] . Result_Bac ← 12.05

NB (treatment of records) : Any operation on the records must be carried out separately: Reading, writing, comparison cannot be done globally, each field must be read, written or compared individually.

Example in C

```
typedef struct date {
int day;
int month;
int year ;
} date;
```

```
Initialization
Date D={0,0,0};
Use
{ D.day = 20;
D.month = 2;
D.year =2020;
}
```

3. Other type definition possibilities: interval type

The interval type is a set of ordered values defined from simple types and characterized by its minimum value (lower end) and its maximum value (upper end).

Declaration syntax

Type Name_Type_Interval = [Val_Min .. Val_Max];
Variable Var_Name: Interval_Type_Name;

Examples

- **Type** Digits = [0 . . 9] ;
- **Type** Letters = ['C' . . 'K'];
- **Type** Note = [0..20];
- **Type** Month = [1..12];

→ We can use the interval type to declare one or more variables.

Variables Note_Analysis, Note_ASD: Note;

The Note_Analyse and Note_ASD variables can take any value from 0 to 20 only.

→ The interval type is a subset of the set of values of an ordinal type. So, all operations possible on the base type are possible on the interval type.

4. Application exercises

Exercise n°1:

Create hour, minute and second interval types, and then a “Time” record that includes these intervals.

Solution

Type Hour = [0..23];

Type Minute = [0..59];

Type Second = [0..59];

Type Time = ***Record***

H: Hour;

M: Minute;

S: second;

EndRecord

Exercise n°2:

Create an array that contains 100 students. Each student is identified by name, first name, result, marks for 9 modules as well as the mention: “admitted”, “adjourned”.

Solution

Type Mention = (admitted, postponed) ; /* enumerated type*/

Type Student = ***Record***

Last name, first name: string;

Marks: array [1 ..9] real;

Result: real;

mention: Mention

EndRecord

Variable T_PV: array [1..100] Student;

Bibliographic References

[1]. AMAD, M. (2016). Algorithmics and Data Structures, Courses and Tutorials. Abderrahmane Mira University of Bejaia.

[2]. BELOUADHA, FZ Algorithms and language C. Mohammed V University – Agdal Mohammadia School of Engineers. IT department

[3]. Berthet, D., & Labatut, V. (2014). Algorithmics & programming in C language vol.2: Practical work topics. Istanbul, Türkiye. Galatasaray University, pp.258.

[4]. Berthet, D., & Labatut, V. (2014). Algorithmics & programming in C-vol language. 1. Istanbul, Türkiye. Galatasaray University, pp.232.

[5]. Berthet, D., & Labatut, V. (2014). Algorithmics & programming in C language vol.3: Answers to practical work. Istanbul, Türkiye. Galatasaray University, pp.217.

[6]. BESSAA, B. (2017). Algorithmic, Exercises with Solutions. https://www.coursehero.com/file/52170520/mi1an-algo-exercises-corriges-1pdf/

[7]. Cormen, TH, Leiserson, CE, Rivest, RL, & Stein, C. (2010). Algorithmics: course with 957 exercises and 158 problems. Dunod.

[8]. Delannoy, C. (1990). Learn to program in Turbo C.

[9]. Delest, M. (2007). Introduction to Algorithmics. Course notes. Bordeaux 1 University.

[10]. Helaoui, M. (2011). Tutorials: Algorithmics and Data Structure. 10.13140/2.1.3800.9601.

[11]. Helaoui, M. (2019). SIDA 2019 2020 v6.pdf. https://www.researchgate.net/publication/337873900_ASDI_2019_2020_v6pdf

[12]. LANGLOIS, Ph. (2013). Programming in C – Exercises. University of Perpignan Via Domitia

[13]. Mohamed, E.M. (2013). Algorithmic. Mohammed V-Agdal University, Faculty of Sciences – Rabat.

[14]. N'Diaye, L., Algo, C., & Sabbar, A. (2007). Algorithmics and data structures.

[15]. Parreaux, J. Lesson 927: Examples of proofs of algorithms: correction and termination.