Fundamentals of Programming Languages

Data Types

Lecture 07

sl. dr. ing. Ciprian-Bogdan Chirila

Lecture outline

- Predefined types
- Programmer defined types
- Scalar types
- Structured data types
  - Cartesian product
  - Finite projection
  - Sequence
  - Recurrence
  - Variable reunions
  - Sets
- Pointer type
- Type Compatibility

Data types

- A set of objects and
- A set of operations to
  - Create
  - Destroy
  - Modify
- Predefined types
  - a certain set of objects specified at language definition
- Unitary construction of objects in advanced PLs
  - structure
  - operations

Predefined Types

- The base of the typing system of a language
- Reflects the system functioning at the hardware level
- Values and operations related to low level data and machine operations

Predefined Types

- Numerical base types
  - int in Algol 68
  - integer in Pascal
  - real in Algol 68 and Pascal
  - float in Ada
  - short int, long int, double
- Mathematical operations
  - +,-,*,/
  - For integers and reals
  - Polymorphic operators - overloaded

Predefined Types

- boolean enumeration type with values
  - true
  - false
- bool in Algol 68
- boolean in Pascal, Java, Ada
- char in Algol 68, Java, Pascal
- character in Ada
- ASCII
- EBCDIC
  - Extended Binary Coded Decimal Interchange Code

Programmer defined types

- The most powerful feature of a typing system is to create new types
- Named
  - type tab=array[1..10] of integer;
- Anonymous
  - var t:array[1..10] of integer;

Scalar types

- Scalar type objects are simple constants which can not be decomposed further
- Integer, real, character, boolean are scalar types
- The programmer can define its own scalar types

Enumeration type

- The user specifies in a list the type values
- type days=(Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday);
- Started in Pascal
- Present in the majority of the PLs

Other scalar types

- Important from the portability point of view
- in Ada
  - type eps is digits 10;
  - a floating point number with a minimum number of 10 significant decimals
- The precision will be preserved independently of the platform

Subdomains

- In Pascal

type working_day=Monday..Friday;

small_caps='a'..'z';

index=0..90;

- In Ada

type eps_1 is new eps range -1.0 .. 1.0;

Structured data types

- PLs offer mechanisms for description and manipulation of data structures containing
  - scalars
  - other structures
- Structuring mechanism
  - features allowing to build structures starting from its components
- Selection mechanism
  - features allowing access to a structure component

Cartesian product

- Structured objects
  - Composed out of a fixed number of components
  - Components are of different types
- The type of the structured objects is the Cartesian product of the sets corresponding to components
- If the types of the components are represented by sets C1, C2, C3, … , Cn
- Each element of structured type will be:

  $T=C1 \times C2 \times \ldots \times Cn$

Cartesian product

- Named also
  - Articles
  - Structures
- In Pascal and Ada - record
- In Algol 68 and C – structure
  - To describe the type of each component
- To select a component means to specify the object and the name of the selected field

Cartesian product example

- Ada

```
type complex is
    record
        re,im:real;
    end record;
-----------------------------
c:complex;
-----------------------------
c.re:=1;
c.im:=0;
-----------------------------
c:=(1,0);
```

Finite projection

- Is a function defined on IT set with values on ET set
- IT – index type
- ET – element type
- var a:array[1..100] of char;
- It is a projection of {1,2,3,…,100} set on the characters set
- The array components called elements are selected through the indexing mechanism
- To the array name we add an index value to select a certain element

Finite projection

- a[k]
  - selects the k index element from array a
  - can be regarded as a application of function a with argument k resulting the value of the element
- In Algol, Ada
  - Selection can be made on a slice not just a single element
  - a[10..19]=(0,1,2,3,4,5,6,7,8,9);

The key moment of binding the set of indexes

- Fixed at compile time
  - Writing the code which establishes the index set
  - Can not be modified during program execution
  - It is the case for Fortran, C, Pascal
- Fixed at run time
  - In the moment of array object creation
  - The size can be unknown at compile time
  - Can depend on program variables
  - It is the case for Algol 60, Basic or Ada
  - In languages with dynamic memory allocation like C pointers are used for dynamic arrays access

The key moment of binding the set of indexes

- Flexible at run time
  - The index set can be modified
  - The size of the array can be modified
  - It is the case for Snobol4 and Algol 68

Sequence

- Is a structure formed out of a random number of components of the same type
- Anytime a component can be added
- Virtually unlimited
- In PLs
  - Strings of characters
  - Sequential file

Sequence

- For strings
  - In PL/I, Ada, Basic, Pascal
  - When declaring a string the maximum length must be known
  - Operations
    - PL dependent
    - Catenation
    - First character selection
    - Last character selection
    - Substring selection
    - etc

Recursion

- A type T is recursive if one of its components is of type T
- Typical examples are
  - Lists
  - Trees
- The objects can have arbitrary shapes and sizes

Recursion

- in pseudocode

```
type node=record
      info : info_type;
      left, right : node;
end;
```

Recursion

- in practice
  - pointers must be used
  - a recursive object of type T must have a reference of a T object
  - not an object itself
  - C, Pascal, Ada, Algol 68
  - In Lisp lists and trees do not need pointers

```
type node=record
    info : info_type;
    left, right : ^node;
end;
```

Variable reunions

- Allow specifying structures which can have several alternatives
- The set of all possible structures represents the reunion of alternative sets

Variable reunions

- In C

```
union {
    float radius;
    float rectangle_sides[2];
    float triangle_sides[3];
} shape;
```

Variable reunions

- at one time shape variable can have only
  - float radius or
  - two float array or
  - three float array
- in an article all fields coexist
- in a union there will be only one of the alternative fields

Variable reunions

- More evolved unions are in Pascal and Ada
- The union is a part of an article with variants

```
type figure=(circle, triangle, rectangle);
shape=record
    length, area:real;
    case shape:figure of
    circle: (radius:real);
    rectangle: (rectangle_sides:array[1..2] of real);
    triangle: (triangle_sides:array[1..3] of real);
end
```

Variable reunions

- Are dangerous
- The correct variant must be used
- All responsibility is left on programmers shoulders (C)
- No compile time checking possible
- No runtime checking possible
- Ada and Pascal cases will be detailed later

Sets

- T is the base type
- Variables of set(T) can have as value any subset generated by values of T including void set
- Operations
  - Reunion
  - Intersection
  - Difference
  - Inclusion tests
  - belonging tests

Sets

- Pascal
  - Has a set type
- When no such mechanism is present
  - Can be implemented by the programmer by
    - Boolean arrays
    - Bit arrays
    - Lists
    - Trees

The pointer type

- A pointer is a reference to an object
- The usual mean to implement recursive data structures
- In C the only way of transmitting parameters by address

Problems with pointers

- Type compatibility violations
- Pseudonyms
- False references

Type compatibility violations

- In PL/I a pointer variable can refer any object
- At compile time is impossible to know the object type and to do appropriate type checking
- runtime checking is possible but they are expensive
- In Pascal, Ada pointers have assigned the object types they may refer
- In C we have the void* generic pointers

Pseudonyms

- The very same object is referred by several names
- Their presence in the code affects its readability

```
var a,b:^t;
a:=new(t);
b:=a;
```

- a and b are pseudonyms

False references

- When a pointer refers an object no longer alive
- Its access is an error

```
var a,b:^t;
a:=new(t);
b:=a;
dispose(a);
```

- b is a false reference even a is set to nil

False references

- In C

```
int *p;
void f()
{
    int x;
    p=&x;
}
f();
```

Type compatibility

- T1 and T2 are compatible types if
  - A value of type T1 can be assigned to a variable of type T2 (and vice versa)
  - A parameter of type T1 corresponds to an actual of type T2 (and vice versa)

Example

```
type
    t=array[1..100] of integer;
    t1=array[1..100] of integer;
    t2=t1;
var
    a,b:array[1..100] of integer;
    c:t;
    d:t;
    e,f:t1;
    g:t2;
```

Theoretical type compatibilities

- Name equivalence
- Structural equivalence

Name equivalence

- when 2 variables
  - declared together or
  - using the same name for the type
- In the example
  - a and b are compatible
  - c and d are compatible
  - e and f are compatible
  - a or b with c or d are not compatible

Structural equivalence

- two variables have compatible types if they have the same structure
- as type checking we will replace the name of the type by its definition
  - recursive process
  - ends when all user defined type are replaced
- Two types are compatible if they have the same description
- In our example
  - a, b, c, d, e, f, g are all compatible

Comparison

- Structural equivalence
  - simplicity of the implementation
- Name equivalence
  - Complex operations in order to determine type compatibility
  - Allows refined abstractions

```
type
    price=integer;
    students_no=integer;
cost:price;
effective:student_no;
```

Comparison

- Variables cost and effective
  - structurally equivalent
  - assigning values from cost to effective or viceversa is a semantic error
- Structural equivalence
  - Algol 68
  - C - structure and union – different types even they have identical structures
- Name equivalence
  - Ada
  - Pascal - equivalence not specified, implementation dependent