

Records

Possibly heterogeneous aggregates where elements are referenced by name

- Difference between a record and an array ?
- Introduced by COBOL.

Structure :

- Example in COBOL :—

```
01 EMP-REC.  
    05 EMP-NAME.  
        10 FIRST      PICTURE IS X(20).  
        10 LAST       PICTURE IS X(20).  
    05 HRLY-RATE PICTURE IS 99V99.
```

- Example in Ada :—

```
EMP_REC :  
    record  
        EMP_NAME :  
            record  
                FIRST :          STRING(1..20);  
                LAST  :          STRING(1..20);  
            end record  
        HRLY_RATE : FLOAT;  
    end record
```

Records

Reference :

- COBOL; Name the field and enclosing record.
 LAST OF EMP-NAME OF EMP-REC
- Most imperative languages use dot notation.
 EMP_REC.EMP_NAME.LAST
- Fully qualified reference
- Elliptical reference

Operations :

- Pascal and Modula-2, records can be assigned.
- Ada, record assignment and comparison.
- Original C, field reference and pointer assignment
- ANSI C and C++, records can be assigned.
- COBOL, MOVE CORRESPONDING statement.

Implementation :

- Fields are stored in adjacent memory.
- Offset address associated with each field.

Records

Variants :- Allow to store different type values at different times during program execution.

- Adv :- Help save/conserves energy.
- DivAdv :- Problem of type checking
- Discriminated union :- Has a tag field or discriminant that tells the current type value.
- Free union :- No tag.

Examples :

- FORTRAN :- EQUIVALENCE statement.
- Pascal, Modula-2 :- tag can be changed without changing the variant part; can omit tag.
- Ada :- tag cannot be changed without changing variant; cannot omit tag; constrained variant variable.
- C and C++ :- no tags; no type checking done.

Implementation :

- Sufficient storage for the largest variant.

Sets

Unordered collections of distinct values. Pascal and Modula-2 provide them.

Examples:

- Modula-2

```
type settype1 = SET OF [blue, green, red];
      settype2 = SET OF [blue, red];
VAR setvar1 = settype1;
```

- Pascal

```
type colors = (blue, green, red);
      colorset = set of colors;
var setvar1 : colorset;
```

Operations :

- union, intersection and equality.

Implementation :

- Stored as bit strings in memory.

Lists

Its an ordered sequence of data structures.

Properties :

- Usually does not have a fixed length.
- Data type of each member may differ.
- List processing languages do not provide explicit attributes of list members.

Example :

- LISP :

Program and data have same form.

(CONS 'A '(B C)) returns (A B C)

(CAR '(A B C)) returns A

(CDR '(A B C)) returns (B C)

Implementation :

- linked-list storage management used.
- head :- first member of list
- tail :- everything other than the head.

Pointers

Variables referencing memory address or NIL. PL/I is the first high-level PL to have pointer variables

Operations :

- Assignment to memory address (with or without allocation).

```
int b;
```

```
int *a = &b;
```

```
int *a = (int *) malloc(sizeof(int));
```

- Reference to value stored in memory cell (Dereference).

```
int b;
```

```
b = *a;
```

Problems :

- Type Checking :- PL/I allowed pointers to point to any type of object.
- Dangling Pointer :- Pointer that contains the address of a dynamic variable that has been deallocated.
- Lost Objects :- Allocated dynamic objects that is no longer accessible to the user program but may contain useful data.

Pointers

Pascal :

- Used to access dynamically allocated variables.
- Dangling pointers can be created by using *dispose*
- *dispose* is very expensive.
 - Ignore *dispose*.
 - Make *dispose* illegal.
 - Deallocate the dynamic variable in question.

Ada :

- Similar to Pascal pointers.
- Implicit deallocation
- Also has `UNCHECKED_DEALLOCATION` function.
- Implicit initialization.

ALGOL 68 :

- Object scope must be at least as large as scope of pointer.
- No explicit deallocation.
- Allows allocation on either heap or stack through keywords.

Implementation of Pointers

Representation :

- Usually 2 or 4 bytes; Need to access all memory locations.

Solution to Dangling Pointer :

- Point to dynamic variable via *tombstones*.
Costly in both time and space. (Macintosh)

Heap Management :

- Reference Counter :- With each cell there is a counter telling how many pointers are pointer to the cell. Pointer is disconnected, decrement counter and check for 0.
 - Space hog; Maintenance time; Circularity.
- Garbage Collection :- Every now and then run through heap and collect all the garbage. Every heap cell has indicator bit.
 - Mark all cells as garbage.
 - All reachable cells are marked not garbage.
 - Reclaim all garbage.