5118006-03 Data Structures

Arrays and Structures

8 Mar 2024

Shin Hong

Topics

2.1 Arrays

2.2 Dynamically Allocated Arrays

2.3 Structures and Unions

2.4 Polynomials

(2.5 Spare matrices)

(2.6 Multidimensional Arrays)

(2.7 Strings)

Array as Abstract Data Type (1/2)

- What vs. How
  - what does an array represent
  - how is an array allocated in memory

- An array is a set of index-value pairs such that each index has a value
  - a correspondence, mapping, partial function
- An array is implemented as a consecutive set of memory locations

Array as Abstract Data Type (2/2)

structure *Array* is

objects: A set of pairs <*index*, *value*> where for each value of *index* there is a value from the set *item*. *Index* is a finite ordered set of one or more dimensions, for example, {0, $\cdots$, $n-1$} for one dimension, {(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)} for two dimensions, etc.

functions:

for all $A \in$ *Array*, $i \in$ *index*, $x \in$ *item*, j, *size* $\in$ integer

Array Create(j, *list*)	::=	**return** an array of j dimensions where *list* is a j-tuple whose ith element is the the size of the ith dimension. *Items* are undefined.
Item Retrieve(A, i)	::=	**if** ($i \in$ *index*) **return** the item associated with index value i in array A **else return** error
Array Store(A,i,x)	::=	**if** (i in *index*) **return** an array that is identical to array A except the new pair <i, x> has been inserted **else return** error.

end *Array*

One-dimensional Arrays in C

```
int list[5] ;
int * plist[5] ;
int * list1 ;
int list2[5] ;
list1 = (int *) malloc(5*sizeof(int)) ;
```

- ex. arrsum.c

Two-dimensional Arrays in C

- An array of arrays
- Ex. 2darr.c
  - `int x[3][5]`
  - `int ** x ;`
  - `int * x[][5] ;`

Structures

- While an array is a finite collection of index-value pairs of the same type, a **structure** is a finite collection of data items where each item is identified by its type and name

- Ex. person.c

Union

- A **union** defines a variable that may be referred as different types depending on the way it is accessed
  - polymorphism

- Ex. union.c

Ordered List

- An ordered list contains a set of item of the same type in sequence
  - ($item_0$, $item_1$, ..., $item_{n-1}$)
  - an empty list is an ordered list

- Operations
  - creating an empty list
  - finding the length of a list
  - reading the items from left to right
  - retrieving the item at the *i*-th position
  - inserting a new item at the *i*-th position
  - deleting the item at the *i*-th position

Ordered List of Integers: Ver. 1

- Ex. intlist.c
  - creating an empty list
  - find the number of the items in the list
  - retrieving an integer at the *i*-th index
  - inserting an integer at the *i*-th index for $0 \leq i \leq n$ where *n* is the number of items
  -