5118006-03 Data Structures

Arrays and Structures

15 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

Team

팀	멤버		
가1		이정환*	
가2	김호진	오승헌	
가3	변소윤	김준홍	
가4	김종엽	김민성	
가5	김은시	BILGUUN	
가6	강지원	홍석진	
가7	이지민	OLEKSII	
가8	곽시열	이민희	
가9	정현준	최현준	
가10	홍희혁	윤나희*	김정모
가11	강영수	채장우	
가12	김민서*	김채영	박하은
가13	남해림	양수진	
가14	김창현	ALI	
가15	김예현	최수민	
가16	이민지	이찬영	
가17	임진형	강승호	김선아*
가18	심민아	구경선	이조희*
가19	백명재	김연휘	

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

Ordered List of Integers: Ver. 1.1

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

Polynomials

- A **polynomial** is a sum of terms where each term has a form of ax^e for x is the variable, a is the coefficient and e is the exponent
  - the largest (lead) exponent is called its degree
  - ex. $A(x) = 3x^{20} + 2x^5 + 4$

- Polynomial operations
  - $A(x) + B\ (x) = \sum(a_i + b_i)x^i$
  - $A(x) \times B\ (x) = \sum(a_i x^i \times (\sum b_j x^j))$

Polynomials: ADT

structure *Polynomial* is

objects: $p(x) = a_1 x^{e_1} + \cdots + a_n x^{e_n}$; a set of ordered pairs of $<e_i, a_i>$ where a_i in *Coefficients* and e_i in *Exponents*, e_i are integers >= 0

functions:

for all *poly*, *poly*1, *poly*2 ∈ *Polynomial*, *coef* ∈ *Coefficients*, *expon* ∈ *Exponents*

Polynomial Zero()	::=	**return** the polynomial, $p(x) = 0$
Boolean IsZero(*poly*)	::=	**if** (*poly*) **return** *FALSE* **else return** *TRUE*
Coefficient Coef(*poly*,*expon*)	::=	**if** (*expon* ∈ *poly*) **return** its coefficient **else return** zero
Exponent Lead_Exp(*poly*)	::=	**return** the largest exponent in *poly*
Polynomial Attach(*poly*, *coef*, *expon*)	::=	**if** (*expon* ∈ *poly*) **return** error **else return** the polynomial *poly* with the term <*coef*, *expon*> inserted
Polynomial Remove(*poly*, *expon*)	::=	**if** (*expon* ∈ *poly*) **return** the polynomial *poly* with the term whose exponent is *expon* deleted **else return** error
Polynomial SingleMult(*poly*, *coef*, *expon*)	::=	**return** the polynomial $poly \cdot coef \cdot x^{expon}$
Polynomial Add(*poly*1, *poly*2)	::=	**return** the polynomial *poly*1 + *poly*2
Polynomial Mult(*poly*1, *poly*2)	::=	**return** the polynomial *poly*1 · *poly*2

end *Polynomial*

Two Different Approaches

- Array-based Approach
  - have an array of doubles, e, such that an element at i-th index, $e[i]$, represents the coefficient of the exponent i.

- Ordered List-based Approach
  - define a structure `term` which consists of an integer for the exponent and a double for the coefficient
  - have an ordered list of terms to represent the containing terms

Polynomials Ver 1

- Ex. poly.c
  - **poly_zero**: creating a zero polynomial
  - **poly_degree**: returning the largest (leading) exponent
  - **poly_coef**: returns the coefficient at a given exponent
  - **poly_attach**: returns the given polynomial after attaching the given term
  - **poly_remove**: return the given polynomial after removing the specified term
  - **poly_show**: print the polynomial to the standard output
  - poly_add: return a new polynomial which is the addition of two given polynomials
  - poly_mut: return a new polynomial which is the product of two given polynomials.