Elementary Data Structures

Bùi Tiến Lên

2024

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Contents

1. Array

2. Linked Lists

3. Variations on Linked List

4. Stack, Queue and Deque

5. Workshop

Array

Array

Concept 1

An **array** is a fixed collection of same-type data that are stored **contiguously** and that are accessible by an **index**.

Concept 2

A **dynamic array** is an array whose size can be changed during the execution of the program.

Example of The sieve of Eratosthenes

- A simple program prints out all prime numbers less than N.

```
void sieve(int N) {
  int i;
  int *a = new int[N];
  for (i = 2; i < N; i++) a[i] = 1;
  for (i = 2; i < N; i++)
    if (a[i])
      for (int j = i; i*j < N; j++) a[i*j] = 0;
  for (i = 2; i < N; i++)
    if (a[i]) cout << " " << i;
  delete[] a;
}
```

- **Challenge**: analysis the program

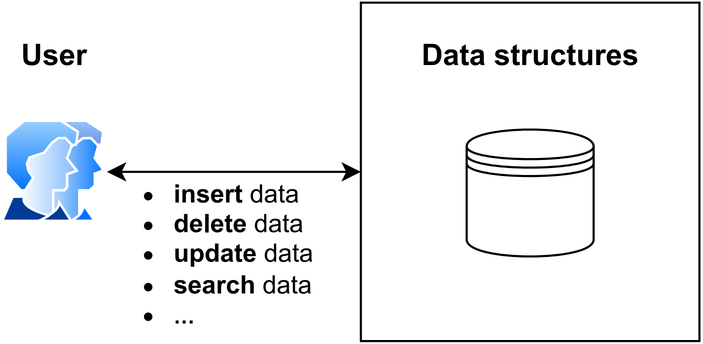
Linked Lists

- Singly Linked Lists
- Ordered Linked List

Data Abstraction

Concept 3

Data abstraction is a process of hiding the implementation details of data structures and operations, while exposing only the essential features and functionalities to the user.

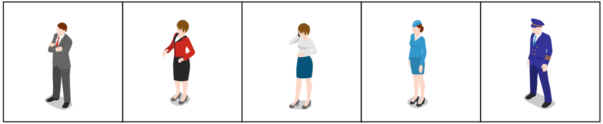


- **insert** data
- **delete** data
- **update** data
- **search** data
- ...

Data Abstraction (cont.)

Data

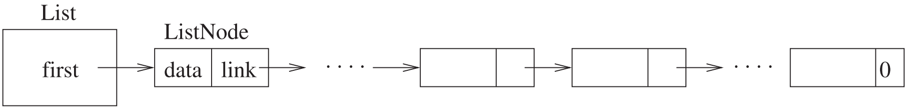
Data structures



Linked Lists

Concept 4

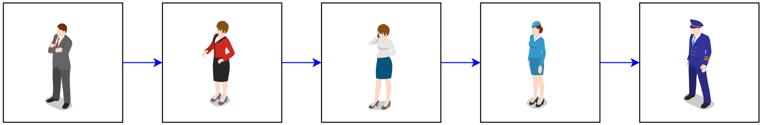
A **linked list** is a set of items where each item is part of a **node** that also contains a **link** to a node. It allows the items be arranged in a linear order.



Linked Lists (cont.)

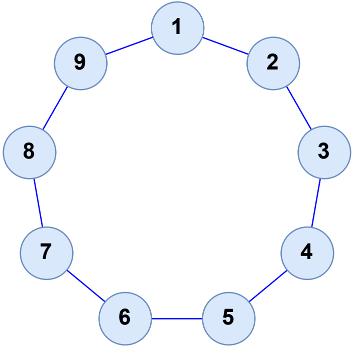
Data

Data structures



Example of Josephus Election

- Imagine that N people have decided to elect a leader by arranging themselves in a circle and eliminating every Mth person around the circle, closing ranks as each person drops out. The problem is to find out which person will be the last one remaining
- If $N = 9$ and $M = 5$



Data Structure for List Node

- We use pointers for links and structures for nodes

```
struct ListNode {
  DataType data;
  ListNode *next;
  ListNode(DataType data, ListNode *next=nullptr) {
    this->data = data;
    this->next = next;
  }
};
typedef ListNode *Link;
```

Create a List Node

- Creating a new node

```
ListNode *p = new ListNode(...);
```

- We so often need to use the phrase “the node referenced by link p” that we simply say “node p”
- It is a null link that points to no node.
- It refers to a dummy node that contains no data.

Delete a List Node

- Deleting a node

```
ListNode *p;
...
delete p;
```

- Writing a function to delete a node

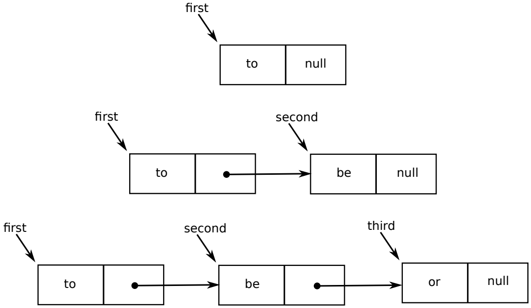
```
void deleteNode(ListNode *p)
{
   ...
}
```

Deep Deletion

- Deleting a node and its link
- Writing a function to delete a node deeply

```
void deepDeleteNode(ListNode *p)
{
    ...
}
```

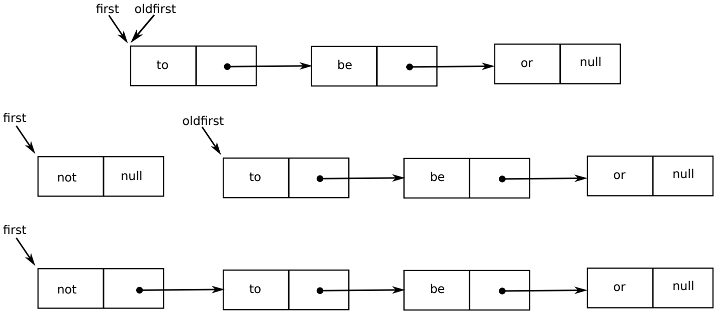
Organize a Linked List



Data Structure for Linked List

```
struct LinkedList {
  ListNode *first; // or ListNode *head;
  LinkedList() {
    this->first = nullptr;
  }
};
```

Insert at The Beginning



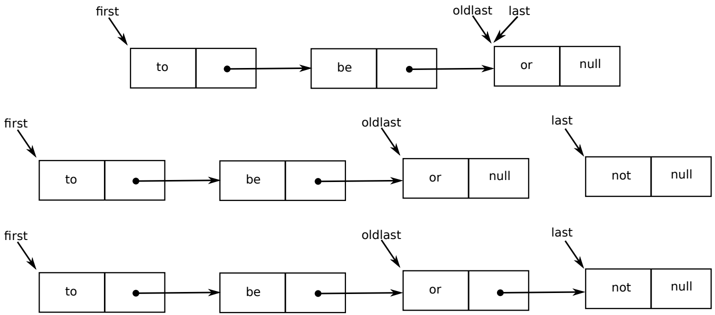
Traversing a Linked List

```
Assign List head to node pointer.
while node pointer is not null
    Display the value member of the node pointed to by node pointer.
    Assign node pointer to its own next member.
end while.
```

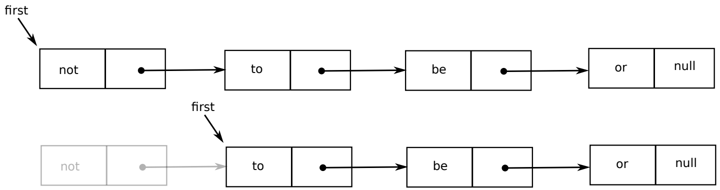
Another Data Structure for Linked List

```
struct LinkedList {
  ListNode *first;
  ListNode *last;
  LinkedList() {
    this->first = nullptr;
    this->last = nullptr;
  }
};
```

Insert at The End



Remove from The Beginning



Search

```
ListNode *search(ListNode *first, DataType k)  {
  ListNode *current = first;
  while(current) {
    if (current->data == k) then return current;
    current = current->next;
  }
  return nullptr;
}
```

Ordered Linked List

Concept 5

An ordered linked list is a data structure that maintains a collection of elements in a linear sequence. The elements in an ordered linked list are arranged in a specific order, such as ascending or descending, based on the values of the elements.

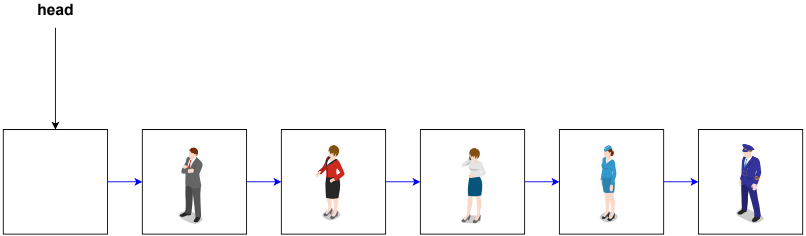
Insert

```
Create a new node.
Store data in the new node.
if there are no nodes in the list
    Make the new node the first node.
else
    Find the first node whose value is greater than or equal to the new
        value, or the end of the list (whichever is first).
    Insert the new node before the found node, or at the end of the list
        if no such node was found.
end if.
```

Dummy Head Node

Concept 6

A dummy head node is a head node that does not store any actual data related to the problem.



Sort

Concept 7

Sorting an unordered linked list is arranging the elements of the list in a specific order, typically ascending or descending

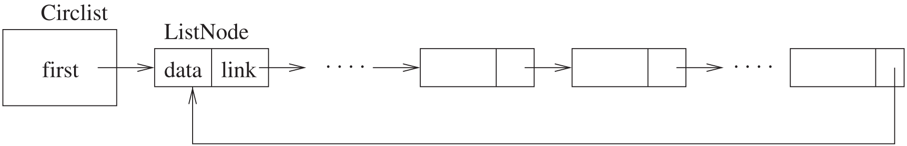
Variations on Linked List

- Circular Linked Lists
- Doubly Linked Lists
- Generalized Lists

Circular Linked Lists

Concept 8

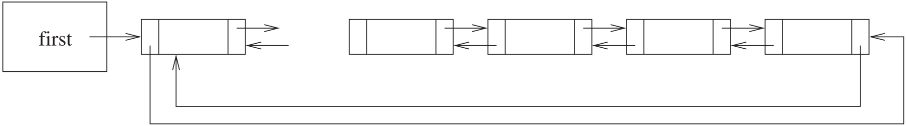
Circular linked list is a variation of linked list in which the last element points to the first element (or the first element points to the last element and).



Doubly Linked Lists

Concept 9

Doubly linked is a variation of linked list in that it has two pointers. One points to the next node as before, while the other points to the previous node.



How to Extend Linked List

We can extend a data structure of linked list by abstracting

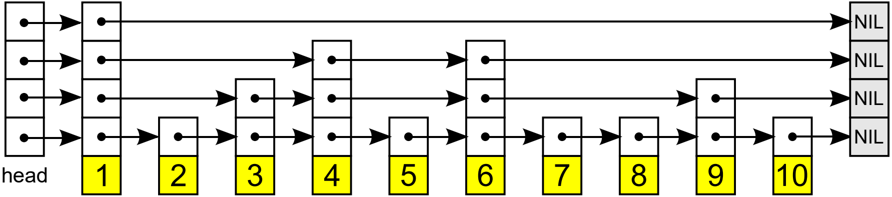
- Data field
- Link field

Multi-Linked List

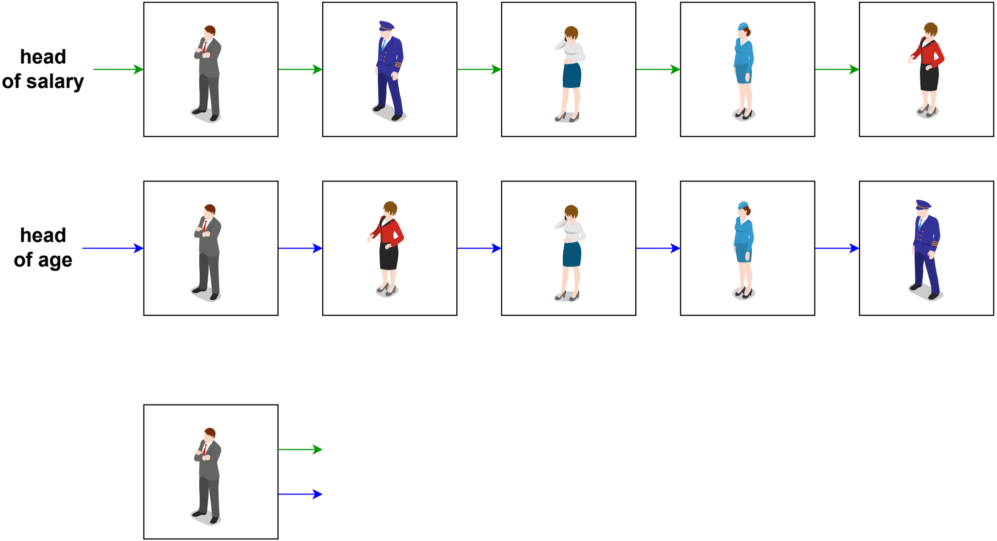
Concept 10

A multi-linked list is a variation of the traditional linked list data structure where each node can have multiple pointers, or references, to other nodes, creating a hierarchical structure.

- Skip List



Multi-Linked List



Generalized Lists

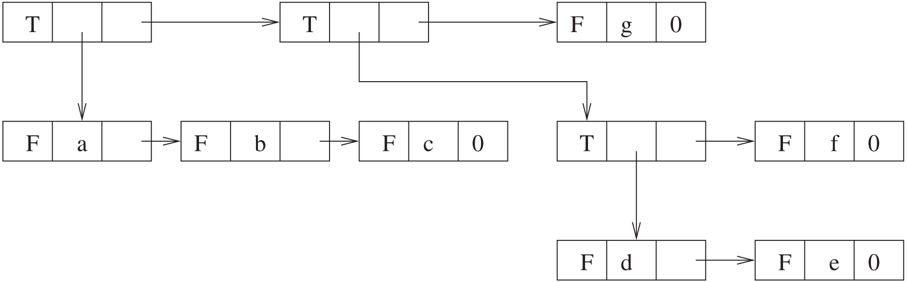
Concept 11

A generalized list l is a finite sequence of $n \geq 0$ elements, $\{e_0, e_1, \ldots, e_{n-1}\}$, where e_i is either an element or a generalized list.

```
struct GenListNode {
  bool tag;
  GenListNode* next;
  union {
    DataType data;
    GenListNode* down;
  };
};
```

Generalized Lists (cont.)

- Consider the generalized list $L = ((a, b, c), ((d, e), f), g)$



Stack, Queue and Deque

- Stack
- Queue
- Deque

Stack

Concept 12

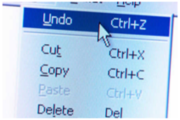
A **stack** is a data structure that stores and retrieves items in a last-in-first- out (LIFO) manner.

Stack API

method	**description**
`boolean isEmpty()`	is the stack empty?
`int size()`	number of items in the stack
`void push(Item item)`	add *item* to the stack
`Item top()`	most recently added item
`void pop()`	remove the most recently added item

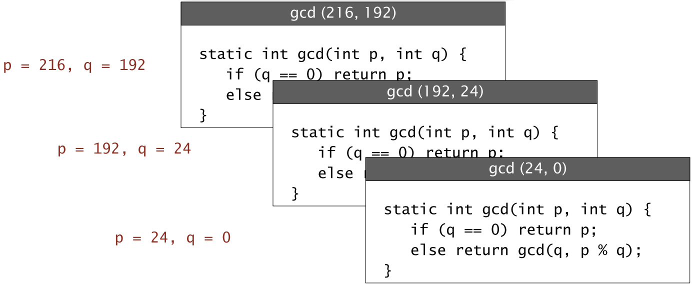
Stack applications

- Parsing in a compiler.
- Java virtual machine.
- Undo in a word processor.
- Back button in a Web browser.
- PostScript language for printers.
- Implementing function calls in a compiler.
- ...



Function calls

- How a compiler implements a function.
  - Function call: **push** local environment and return address.
  - Return: **pop** return address and local environment.



p = 216, q = 192

gcd (216, 192)

```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else
}
```

p = 192, q = 24

gcd (192, 24)

```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else
}
```

p = 24, q = 0

gcd (24, 0)

```
static int gcd(int p, int q) {
   if (q == 0) return p;
   else return gcd(q, p % q);
}
```

Remove recursion

- **Recursive function**: Function that calls itself.
- Can always use an explicit stack to remove recursion.
- **Challenge**: reimplement quicksort without using recursion

Array
Linked Lists
Singly Linked Lists
Ordered Linked List
Variations on Linked List
Circular Linked Lists
Doubly Linked Lists
Generalized Lists
Stack, Queue and Deque
Stack
Queue
Deque
Workshop

Arithmetic expression

Arithmetic expression can be represented in

- infix

  `<operand 1>`**`<operator>`**`<operand 2>`

- prefix (Polish Notation)

  `<operator>``<operand 1><operand 2>`

- postfix (Reverse-Polish Notation)

  `<operand 1><operand 2>`**`<operator>`**

infix	**prefix**	**postfix**
A+B*C	+*BCA	BC*A+
(A-B)/C	/-ABC	AB-C/
(A+B)*(C-D)	*+AB-CD	AB+CD-*

Conversion of an infix expression to postfix

- Convert infixExp to postfixExp

```
stackOps.push('(')
infixExp.append(')')
while not infixExp.end()?
    tok ← infixExp.nextToken()
    if tok is operand then postfixExp.append(tok)
    if tok is “(” then stackOps.push(tok)
    if tok is operator then
        while precedence of stackOps.top() is higher than or equal tok?
            postfixExp.append(stackOps.pop())
        stackOps.push(tok)
    if tok is “)” then
        while stackOps.top() is not “(”?
            postfixExp.append(stackOps.pop())
    stackOps.pop()
```

Example

- Convert the infix expression `(A+B)*(C-(D+A))` into a postfix expression

stackOps	infixExp	postfixExp
(	(A+B)*(C-(D+A)))	
(	A+B)*(C-(D+A)))	
(	+B)*(C-(D+A)))	
(	B)*(C-(D+A)))	
(	)*(C-(D+A)))	
(	*(C-(D+A)))	
(	(C-(D+A)))	
(	C-(D+A)))	
(	-(D+A)))	
(	(D+A)))	
(	D+A)))	
(	+A)))	
(	A)))	
(	)))	
(	))	
(	)	

Arithmetic expression evaluation

- A *simple version* of two-stack algorithm proposed by E. W. Dijkstra

Scan tokens from the expression (fully parenthesized)

If token

- **Value**: push onto the **value stack**.
- **Operator**: push onto the **operator stack**.
- **Left parenthesis**: ignore.
- **Right parenthesis**:
  - pop operator and two values.
  - push the result of applying that operator to those values onto the operand stack.

Arithmetic expression evaluation (cont.)

- Evaluate the expression (1 + ((2 + 3) * (4 * 5)))

value stack	operator stack	
		(1 + ((2 + 3) * (4 * 5)))
1		+ ((2 + 3) * (4 * 5)))
1	+	((2 + 3) * (4 * 5)))
1 2	+	+ 3) * (4 * 5)))
1 2	+ +	3) * (4 * 5)))
1 2 3	+ +	) * (4 * 5)))
1 5	+	* (4 * 5)))
1 5	+ *	(4 * 5)))
1 5 4	+ *	* 5)))
1 5 4	+ * *	5)))
1 5 4 5	+ * *	)))
1 5 20	+ *	))
1 100	+	)
101		

Implementation (simple)

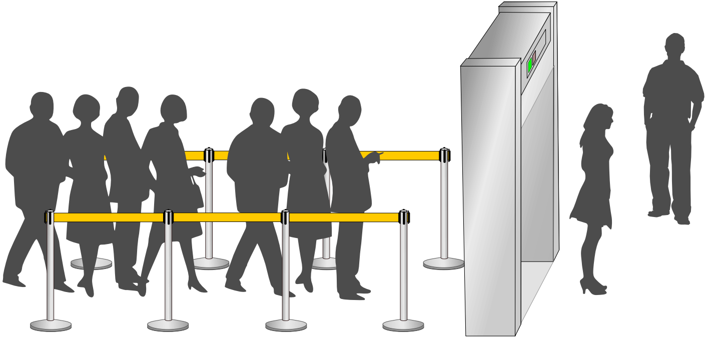
- Input **in** is a arithmetic expression that is fully parenthesized and contains delimiters (space characters)

```
double evaluate(istream& in) {
  stack<string> ops;
  stack<double> vals;
  string tok;
  while (!in.eof()) {
    in >> tok;
    if (tok == "(");
    else if (tok == "+" || tok == "*") ops.push(tok);
    else if (tok == ")") {
      string op = ops.top(); ops.pop();
      double val2 = vals.top(); vals.pop();
      double val1 = vals.top(); vals.pop();
      if (op == "+") vals.push(val1 + val2);
      else if (op == "*") vals.push(val1 * val2);
    }
    else vals.push(stod(tok));
  }
  return vals.top();
}
```

Queue

Concept 13

A **queue** is a data structure that stores and retrieves items in a first-in- first-out (FIFO) manner.



Queue API

method	description
`boolean isEmpty()`	is the queue empty?
`int size()`	number of items in the queue
`void enqueue(Item item)`	add *item* to the queue
`void dequeue()`	remove the least recently added item
`Item front()`	the least recently added item

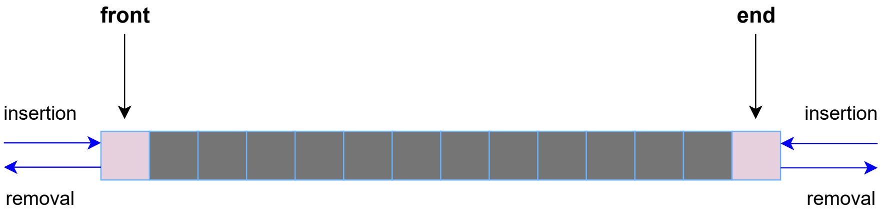
Queue applications

- Operating systems (queuing messages, IO requests, mouse movements, etc),
- Web servers (queuing incoming requests, file operations, etc)
- Ticket counter line where people who come first will get his ticket first
- Bank line where people who come first will done his transaction first
- ...

Deque

Concept 14

The **deque** stands for **Double Ended Queue**. Deque is a linear data structure where the insertion and deletion operations are performed from both ends. We can say that deque is a generalized version of the queue.



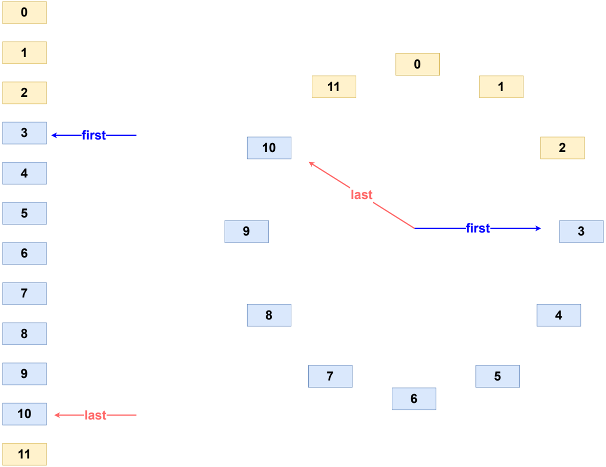
Deque (cont.)

Some restricted deques

- If we insert at the end and remove at the end, we get a stack
- if we insert at the end and remove at the beginning, we get a FIFO queue

Deque (cont.)

Array-based deque



Deque API

method	description
`void push_front(Item item)`	Insert item at the front
`void push_back(Item item)`	Insert item at the back
`void pop_front()`	Remove at the front
`void pop_back()`	Remove at the back
...	

Workshop

Quiz

1. What is a linked list?

   ....

   ....

   ....

2. What is a stack?

   ....

   ....

   ....

3. What is a queue?

   ....

   ....

   ....

Quiz (cont.)

4. A letter means push and an asterisk means pop in the sequence

   `EAS*Y*QUE***ST***IO*N***`

   Give the sequence of values returned by the pop operations.

5. An uppercase letter means put at the beginning, a lowercase letter means put at the end, a plus sign means get from the beginning, and an asterisk means get from the end in the sequence

   `EAs+Y+QUE**+st+*+IO*n++*`

   Give the sequence of values returned by the get operations when this sequence of operations is performed on an initially empty deque.

⌨ Projects

1. Design and implement class `Polynomial`
2. Design and implement class `Tensor`
3. Design and implement class `SparseMatrix`
4. Design and implement class `Expression`

References

Deitel, P. (2016).
C++: How to program.
Pearson.

Gaddis, T. (2014).
Starting Out with C++ from Control Structures to Objects.
Addison-Wesley Professional, 8th edition.

Jones, B. (2014).
Sams teach yourself C++ in one hour a day.
Sams.