Δομές Δεδομένων και Αλγόριθμοι - Εργαστήριο 5
Στοίβες (stacks) και ουρές (queues), οι δομές στοίβα και ουρά στην STL

Τ.Ε.Ι. Ηπείρου, Τμήμα Μηχανικών Πληροφορικής Τ.Ε.
Χρήστος Γκόγκος - Αναπληρωτής Καθηγητής

1 Εισαγωγή

Οι στοίβες και οι ουρές αποτελούν απλές δομές δεδομένων που είναι ιδιαίτερα χρήσιμες στην επίλυση αλγοριθμικών προβλημάτων. Η στοίβα είναι μια λίστα στοιχείων στην οποία τα νέα στοιχεία τοποθετούνται στην κορυφή και όταν πρόκειται να αφαιρεθεί ένα στοιχείο αυτό πάλι συμβαίνει από την κορυφή των στοιχείων της στοίβας. Από την άλλη μεριά, η ουρά είναι επίσης μια λίστα στοιχείων στην οποία όμως οι εισαγωγές γίνονται στο πίσω άκρο της ουράς ενώ οι εξαγωγές πραγματοποιούνται από το εμπρός άκρο της ουράς. Στο εργαστήριο αυτό θα παρουσιαστούν υλοποιήσεις της στοίβας και της ουράς. Επιπλέον, θα παρουσιαστούν οι δομές της STL std::stack και std::queue. Ο κώδικας όλων των παραδειγμάτων βρίσκεται στο https://github.com/chgogos/ceteiep_dsa.

2 Στοίβα

Η στοίβα (stack) είναι μια ειδική περίπτωση γραμμικής λίστας στην οποία οι εισαγωγές και οι διαγραφές επιτρέπονται μόνο από το ένα άκρο. Συνήθως αυτό το άκρο λέγεται κορυφή (top). Πρόκειται για μια δομή στην οποία οι εισαγωγές και οι εξαγωγές γίνονται σύμφωνα με τη μέθοδο τελευταίο μέσα πρώτο έξω (LIFO=Last In First Out).

Στον κώδικα 1 παρουσιάζεται μια υλοποίηση στοίβας που χρησιμοποιεί για την αποθήκευση των στοιχείων της έναν πίνακα. Εναλλακτικά, στη θέση του πίνακα μπορεί να χρησιμοποιηθεί συνδεδεμένη λίστα. Μια υλοποίηση στη γλώσσα C μπορεί να βρεθεί στην αναφορά [2], ενώ στην εργασία [1] παρουσιάζονται 16(!) διαφορετικοί τρόποι υλοποίησης της στοίβας στην C++.

Στο παράδειγμα που ακολουθεί ωθούνται σε μια στοίβα τα γράμματα της αγγλικής αλφαβήτου (A-Z) και στη συνέχεια απωθούνται ένα προς ένα και μέχρι η στοίβα να αδειάσει.

```
1  #include <iostream>
2
3  using namespace std;
4
5  template <typename T> class my_stack {
6  private:
7    T *data;
8    int top, capacity;
9
10 public:
11   // constructor
12   my_stack(int c) {
13     top = -1;
14     capacity = c;
15     data = new T[capacity];
16   }
```

```
17
18    // destructor
19    ~my_stack() { delete[] data; }
20
21    bool empty() { return (top == -1); }
22
23    void push(T elem) {
24      if (top == (capacity - 1))
25        throw "The stack is full";
26      else {
27        top++;
28        data[top] = elem;
29      }
30    }
31
32    T pop() {
33      if (top == -1)
34        throw "the stack is empty";
35      top--;
36      return data[top + 1];
37    }
38
39    void print() {
40      for (int i = 0; i <= top; i++)
41        cout << data[i] << " ";
42      cout << endl;
43    }
44  };
45
46  int main() {
47    cout << "Custom stack implementation" << endl;
48    my_stack<char> astack(100);
49    for (char c = 65; c < 65 + 26; c++)
50      astack.push(c);
51    astack.print();
52    while (!astack.empty())
53      cout << astack.pop() << " ";
54    cout << endl;
55  }
```

Κώδικας 1: Υλοποίηση στοίβας (stack_oo.cpp)

```
1  Custom stack impementation
2  A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
3  Z Y X W V U T S R Q P O N M L K J I H G F E D C B A
```

3 Ουρά

Η ουρά (queue) είναι μια ειδική περίπτωση γραμμικής λίστας στην οποία επιτρέπονται εισαγωγές στο πίσω άκρο της και εξαγωγές από το εμπρός άκρο της μόνο. Τα δύο αυτά άκρα συνήθως αναφέρονται ως πίσω (rear) και εμπρός (front) αντίστοιχα. Η ουρά είναι μια δομή στην οποία οι εισαγωγές και οι εξαγωγές γίνονται σύμφωνα με τη μέθοδο πρώτο μέσα πρώτο έξω (FIFO=First In First Out).

Στη συνέχεια παρουσιάζεται μια υλοποίηση ουράς στην οποία τα δεδομένα της τοποθετούνται σε έναν πίνακα (εναλλακτικά θα μπορούσε να είχε χρησιμοποιηθεί μια άλλη δομή όπως για παράδειγμα η συνδεδεμένη λίστα). Ο πίνακας λειτουργεί κυκλικά, δηλαδή όταν συμπληρωθεί και εφόσον υπάρχουν διαθέσιμες κενές θέσεις στην αρχή του πίνακα, τα νέα στοιχεία που πρόκειται να εισαχθούν στην ουρά τοποθετούνται στις πρώτες διαθέσιμες, ξεκινώντας από την αρχή του πίνακα, θέσεις.

```
#include <iostream>

using namespace std;

template <typename T> class my_queue {
private:
  T *data;
  int front, rear, capacity, size;

public:
  // constructor
  my_queue(int c) {
    front = 0;
    rear = -1;
    size = 0;
    capacity = c;
    data = new T[capacity];
  }

  // destructor
  ~my_queue() { delete[] data; }

  bool empty() { return (size == 0); }

  void enqueue(T elem) {
    if (size == capacity)
      throw "The queue is full";
    else {
      rear++;
      rear %= capacity;
      data[rear] = elem;
      size++;
    }
  }

  T dequeue() {
    if (size == 0)
      throw "the queue is empty";
    T x = data[front];
    front++;
    front %= capacity;
    size--;
    return x;
  }


  void print(bool internal = true) {
    for (int i = front; i < front + size; i++)
      cout << data[i % capacity] << " ";
    cout << endl;
    if (internal){
      for (int i = 0; i < capacity; i++)
        if (front <= rear && i >= front && i <= rear)
          cout << "[" << i << "]->" << data[i] << " ";
        else if (front >= rear && (i >= front || i <= rear))
          cout << "[" << i << "]->" << data[i] << " ";
        else
          cout << "[" << i << "]->X ";
    }
    cout << " (front:" << front << " rear:" << rear << ")" << endl;
```

```
61    }
62 };
63
64 int main() {
65    cout << "Custom queue implementation" << endl;
66    my_queue<int> aqueue(10);
67    cout << "1. Enqueue 10 items" << endl;
68    for (int i = 51; i <= 60; i++)
69       aqueue.enqueue(i);
70    aqueue.print();
71    cout << "2. Dequeue 5 items" << endl;
72    for (int i = 0; i < 5; i++)
73       aqueue.dequeue();
74    aqueue.print();
75    cout << "3. Enqueue 3 items" << endl;
76    for (int i = 61; i <= 63; i++)
77       aqueue.enqueue(i);
78    aqueue.print();
79 }
```

Κώδικας 2: Υλοποίηση ουράς (queue_oo.cpp)

```
1  Custom queue implementation
2  1. Enqueue 10 items
3  51 52 53 54 55 56 57 58 59 60
4  [0]->51 [1]->52 [2]->53 [3]->54 [4]->55 [5]->56 [6]->57 [7]->58 [8]->59 [9]->60 (front:0 rear:9)
5  2. Dequeue 5 items
6  56 57 58 59 60
7  [0]->X [1]->X [2]->X [3]->X [4]->X [5]->56 [6]->57 [7]->58 [8]->59 [9]->60 (front:5 rear:9)
8  3. Enqueue 3 items
9  56 57 58 59 60 61 62 63
10 [0]->61 [1]->62 [2]->63 [3]->X [4]->X [5]->56 [6]->57 [7]->58 [8]->59 [9]->60 (front:5 rear:2)
```

4 Οι δομές στοίβα και ουρά στην STL

Οι δομές std::stack και std::queue έχουν υλοποιηθεί στην STL ως container adaptors δηλαδή κλάσεις που χρησιμοποιούν εσωτερικά ένα άλλο container και παρέχουν ένα συγκεκριμένο σύνολο από λειτουργίες που επιτρέπουν την προσπέλαση και την τροποποίηση των στοιχείων τους.

4.1 std::stack

Το προκαθορισμένο εσωτερικό container που χρησιμοποιεί η std::stack είναι το std::deque. Ωστόσο, μπορούν να χρησιμοποιηθούν και τα std::vector και std::list καθώς και τα τρία αυτά containers παρέχουν τις λειτουργίες empty(), size(), push_back(), pop_back() και back() που απαιτούνται για να υλοποιηθεί το stack interface [3]. Τυπικές λειτουργίες που παρέχει η std::stack είναι οι ακόλουθες:

- empty(), ελέγχει αν η στοίβα είναι άδεια.
- size(), επιστρέφει το μέγεθος της στοίβας.
- top(), προσπελαύνει το στοιχείο που βρίσκεται στη κορυφή της στοίβας (χωρίς να το αφαιρεί).
- push(), ωθεί ένα στοιχείο στη κορυφή της στοίβας
- pop(), αφαιρεί το στοιχείο που βρίσκεται στη κορυφή της στοίβας.

Ένα παράδειγμα χρήσης της std::stack παρουσιάζεται στη συνέχεια.

```
1 #include <deque>
2 #include <iostream>
3 #include <list>
4 #include <stack>
```

```cpp
#include <vector>

using namespace std;
int main(void) {
   cout << "std::stack example" << endl;
   stack<char> items; // adaptor over a deque container
   // stack<char, deque<char>> items; // adaptor over a deque container
   // stack<char, vector<char>> items; // adaptor over a vector container
   // stack<char, list<char>> items; // adaptor over a list container

   for (char c = 65; c < 65 + 26; c++) {
      cout << c << " ";
      items.push(c);
   }
   cout << endl;

   while (!items.empty()) {
      cout << items.top() << " ";
      items.pop();
   }
   cout << endl;
}
```

Κώδικας 3: Παράδειγμα χρήσης της std::stack (stl_stack_example.cpp)

```
std::stack example
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Z Y X W V U T S R Q P O N M L K J I H G F E D C B A
```

4.2 std::queue

Στην περίπτωση του std::queue το εσωτερικό container μπορεί να είναι κάποιο από τα containers std::dequeue, std::list (προκαθορισμένη επιλογή) ή οποιοδήποτε container που υποστηρίζει τις λειτουργίες empty(), size(), front(), back(), push_back() και pop_front() [4]. Τυπικές λειτουργίες που παρέχει η std::queue είναι οι ακόλουθες:

- empty(), ελέγχει αν η ουρά είναι άδεια.
- size(), επιστρέφει το μέγεθος της ουράς.
- front(), προσπελαύνει το στοιχείο που βρίσκεται στο εμπρός άκρο της ουράς (χωρίς να το αφαιρεί).
- back(), προσπελαύνει το στοιχείο που βρίσκεται στο πίσω άκρο της ουράς (χωρίς να το αφαιρεί).
- push(), εισάγει ένα στοιχείο στο πίσω άκρο της ουράς
- pop(), εξάγει το στοιχείο που βρίσκεται στο εμπρός άκρο της ουράς.

Ένα παράδειγμα χρήσης της std::queue παρουσιάζεται στη συνέχεια.

```cpp
#include <iostream>
#include <queue>
#include <list>

using namespace std;

int main() {
   cout << "std::queue" << endl;
   queue<int> aqueue; // adaptor over a deque container
   // queue<int, deque<int>> aqueue; // adaptor over a deque container
   // queue<int, list<int>> aqueue; // adaptor over a list container

   cout << "1. Enqueue 10 items" << endl;
```

```
14    for (int i = 51; i < 60; i++) {
15       cout << i << " ";
16       aqueue.push(i);
17    }
18    cout << endl << "2. Dequeue 5 items" << endl;
19    for (int i = 0; i < 5; i++) {
20       cout << aqueue.front() << " ";
21       aqueue.pop();
22    }
23    cout << endl << "3. Enqueue 3 items" << endl;
24    for (int i = 61; i <= 63; i++) {
25       cout << i << " ";
26       aqueue.push(i);
27    }
28    while (!aqueue.empty()) {
29       cout << aqueue.front() << " ";
30       aqueue.pop();
31    }
32    cout << endl;
33 }
```

Κώδικας 4: Παράδειγμα χρήσης της std::queue (stl_queue_example.cpp)

```
1 std::queue
2 1. Enqueue 10 items
3 51 52 53 54 55 56 57 58 59
4 2. Dequeue 5 items
5 51 52 53 54 55
6 3. Enqueue 3 items
7 61 62 63 56 57 58 59 61 62 63
```

5 Παραδείγματα

5.1 Παράδειγμα 1

Να γραφεί πρόγραμμα που να δέχεται μια φράση ως παράμετρο γραμμής εντολών (command line argument) και να εμφανίζει το εάν είναι παλινδρομική ή όχι. Μια φράση είναι παλινδρομική όταν διαβάζεται η ίδια από αριστερά προς τα δεξιά και από δεξιά προς τα αριστερά.

```
1  #include <iostream>
2  #include <stack>
3
4  using namespace std;
5  // examples of palindromic sentences:
6  // SOFOS, A MAN A PLAN A CANAL PANAMA, AMORE ROMA, LIVE NOT ON EVIL
7  int main(int argc, char **argv) {
8     if (argc != 2) {
9        cerr << "Usage examples: " << endl;
10       cerr << "\t\t " << argv[0] << " SOFOS" << endl;
11       cerr << "\t\t " << argv[0] << " \"A MAN A PLAN A CANAL PANAMA\"" << endl;
12       exit(-1);
13    }
14    string str = argv[1];
15    stack<char> astack;
16    string str1;
17    for (char c : str)
18       if (c != ' ') {
19          str1 += c;
20          astack.push(c);
21       }
```

```
22    string str2;
23    while (!astack.empty()) {
24       str2 += astack.top();
25       astack.pop();
26    }
27    if (str1 == str2)
28       cout << "The sentence " << str << " is palindromic." << endl;
29    else
30       cout << "The string " << str << " is not palindromic." << endl;
31 }
```

Κώδικας 5: Έλεγχος παλινδρομικής φράσης (lab05_ex1.cpp)

```
1  $ ./lab05_ex1
2  Usage examples:
3                    ./lab05_ex1 SOFOS
4                    ./lab05_ex1 "A MAN A PLAN A CANAL PANAMA"
5
6  $ ./lab05_ex1 "A MAN A PLAN A A CANAL PANAMA"
7  The string A MAN A PLAN A A CANAL PANAMA is not palindromic.
8
9  $ ./lab05_ex1 "A MAN A PLAN A A CANAL PANAM"
10 The string A MAN A PLAN A A CANAL PANAM is not palindromic.
```

5.2 Παράδειγμα 2

Να γραφεί πρόγραμμα που να δέχεται ένα δυαδικό αριθμό ως λεκτικό και να εμφανίζει την ισοδύναμη δεκαδική του μορφή.

```
1  #include <iostream>
2  #include <stack>
3  #include <string> // stoi
4
5  using namespace std;
6  int main() {
7     string bs;
8     stack<int> astack;
9     cout << "Enter a binary number: ";
10    cin >> bs;
11    for (char c : bs) {
12       if (c != '0' && c != '1') {
13          cerr << "use only digits 0 and 1" << endl;
14          exit(-1);
15       }
16       astack.push(c - '0');
17    }
18
19    int sum = 0, x = 1;
20    while (!astack.empty()) {
21       sum += astack.top() * x;
22       astack.pop();
23       x *= 2;
24    }
25    cout << "Decimal: " << sum << endl;
26
27    cout << "Decimal: " << stoi(bs, nullptr, 2) << endl; // one line solution :)
28 }
```

Κώδικας 6: Μετατροπή δυαδικού σε δεκαδικό (lab05_ex2.cpp)

```
1 Enter a binary number: 10101010101010101111111100111
2 Decimal: 178958311
3 Decimal: 178958311
```

6 Ασκήσεις

1. Να υλοποιηθεί η δομή της ουράς χρησιμοποιώντας αντικείμενα στοίβας (std::stack) και τις λειτουργίες που επιτρέπονται σε αυτά. Υλοποιήστε τις λειτουργίες της ουράς empty(), size(), enqueue(), dequeue() και front().

2. Να υλοποιηθεί η δομή της στοίβας χρησιμοποιώντας αντικείμενα ουράς (std::queue) και τις λειτουργίες που επιτρέπονται σε αυτά. Υλοποιήστε τις λειτουργίες της στοίβας empty(), size(), push(), pop() και top().

Αναφορές

[1] Sixteen Ways To Stack a Cat, by Bjarne Stroustrup http://www.stroustrup.com/stack_cat.pdf

[2] Tech Crash Course, C Program to Implement a Stack using Singly Linked List, http://www.techcrashcourse.com/2016/06/c-program-implement-stack-using-linked-list.html

[3] C++ Reference Material by Porter Scobey, The STL stack Container Adaptor http://cs.stmarys.ca/ porter/csc/ref/stl/cont_stack.html

[4] C++ Reference Material by Porter Scobey, The STL queue Container Adaptor http://cs.stmarys.ca/ porter/csc/ref/stl/cont_queue.html