КПІ ім. Ігоря Сікорського

Інститут прикладного системного аналізу

Кафедра Системного проектування

Алгоритми та структури даних

**Лабораторна робота №4**

**«Дослідження структур даних стек та черга»**

Виконав:

Студент групи ДА-02

Рудік Андрій

Варіант №16

Мета роботи:

Ознайомитись і дослідити принципи FIFO, LIFO та лінійні структури даних: стек та чергу. Реалізувати їх на основі статичних масивів, набути навичок роботи зі структурами та вказівниками на мові програмування С/C++.

Варіант №16.

Завдання №1

Структура даних Книга має наступні поля: назва, кількість сторінок,

ціна. Виконати наступні завдання:

1) Створити стопку (стек) книг, розмір стопки 6.

2) Додати в неї 5 різних книг.

3) Забрати 2 верхні книги.

4) Додати ще 4 нові книги у стопку.

5) Взяти верхню книгу, зменшити кількість її сторінок на 10 та повернути

у стопку.

6) Забрати 3 книги зі стопки.

7) Забрати ще 5 книг зі стопки.

[Результат](https://github.com/franmajka/algorithms-and-data-structures/blob/master/laba%234/res/stack.jpg)

Лістинг

#include <iostream>

#include <string>

#include <cstdlib>

#define STACK\_SIZE 6

#define NUMBER\_WIDTH 3

#define TITLE\_WIDTH 40

#define NUMBER\_OF\_PAGES\_WIDTH 10

#define AUTHOR\_WIDTH 30

using *namespace* std;

*enum* Errors {

  BAD\_FORMAT,

  STACK\_UNDERFLOW,

  STACK\_OVERFLOW,

};

*enum* Actions {

  EXIT = -1,

  STACK\_ACTIONS,

  BOOK\_ACTIONS,

  ADDING\_BOOK,

};

*template* <*typename* T>

*struct* ArrayWithLength {

  T \*arr;

  size\_t length;

  ArrayWithLength(size\_t *length\_*) {

    length = *length\_*;

    arr = new T[*length\_*];

  }

  ~ArrayWithLength() {

    delete[] arr;

  }

};

*template* <*typename* T>

*void* realloc(ArrayWithLength<T> \**arrS*, *int* *change*) {

*int* length = *change* < 0 ? *arrS*->length + *change* : *arrS*->length;

  if (length < 0) throw out\_of\_range("Новая длина массива меньше 0");

  T \*newStrArr = new T[*arrS*->length + *change*];

  for (*int* i = 0; i < length; i++) {

    newStrArr[i] = *arrS*->arr[i];

  }

  delete[] *arrS*->arr;

*arrS*->arr = newStrArr;

*arrS*->length += *change*;

}

// Удаляет символы справа и слева

*void* strTrim(string &*str*, *char* *c* = ' ') {

  while (*str*[0] == *c*) {

*str*.erase(0, 1);

  }

  size\_t length = *str*.length();

  while (length && *str*[length - 1] == *c*) {

*str*.erase(length - 1, length);

    length = *str*.length();

  }

}

// Принимает строку и разделить

// Возвращает массив с длиной, состоящий из подстрок, на которые была поделена изначальная строка разделителем

ArrayWithLength<string>\* strSplit(string *str*, string *delimiter* = " ") {

  ArrayWithLength<string> \*arrS = new ArrayWithLength<string>(1);

  size\_t pos;

  string tmp;

  while ((pos = *str*.find(*delimiter*)) != string::npos) {

    tmp = *str*.substr(0, pos);

*str*.erase(0, pos + *delimiter*.length());

    arrS->arr[arrS->length - 1] = tmp;

    realloc(arrS, 1);

  }

  arrS->arr[arrS->length - 1] = *str*;

  return arrS;

}

*struct* Book {

  string title;

*int* numberOfPages;

  string author;

  // Принимает строку в формате "Н | КС | А"

  // В случае любой возможной ошибки вызывает BAD\_FORMAT

  Book(string *input*) {

    ArrayWithLength<string> \*arrS = strSplit(*input*, "|");

    if (arrS->length != 3) {

      delete arrS;

      throw BAD\_FORMAT;

    }

    for (size\_t i = 0; i < arrS->length; i++) {

      strTrim(arrS->arr[i], ' ');

    }

    title = arrS->arr[0];

    try {

      numberOfPages = stoi(arrS->arr[1]);

      if (numberOfPages <= 0) throw out\_of\_range("Не положительное значение страниц");

    } catch (...) {

      delete arrS;

      throw BAD\_FORMAT;

    }

    author = arrS->arr[2];

    delete arrS;

  }

  ~Book() {

    // Не представляю что тут удалять...

    // По идее структура объявлена динамически и её неплохо бы удалить но удалять нечего так-то

  }

};

*void* printBook(Book \**book*) {

  cout << "Название: " << *book*->title << endl << "Количество страниц: " << *book*->numberOfPages << endl << "Автор: " << *book*->author << endl;

}

*template* <*typename* T>

*struct* Stack {

  T \*\*arr;

  size\_t size;

*int* topIndex = -1;

  Stack() {

    size = STACK\_SIZE;

    arr = new T\*[STACK\_SIZE];

  }

  Stack(size\_t *size\_*) {

    size = *size\_*;

    arr = new T\*[*size\_*];

  }

  ~Stack() {

    for (*int* i = 0; i <= topIndex; i++) {

      delete arr[i];

    }

    delete[] arr;

  }

};

*template* <*typename* T>

*bool* isEmpty(Stack<T> \**stack*) {

  return *stack*->topIndex == -1;

}

*template* <*typename* T>

size\_t getSize(Stack<T> \**stack*) {

  return *stack*->topIndex + 1;

}

*template* <*typename* T>

T\* getTop(Stack<T> \**stack*) {

  if (isEmpty(*stack*)) {

    throw STACK\_UNDERFLOW;

  }

  return *stack*->arr[*stack*->topIndex];

}

// Возвращает элемент, что был удален, в случае если это возможно и выбрасывает ошибку в остальных случаях.

// Уменьшает значение topIndex на 1.

*template* <*typename* T>

T\* pop(Stack<T> \**stack*) {

  if (isEmpty(*stack*)) {

    throw STACK\_UNDERFLOW;

  }

  T \*el = *stack*->arr[*stack*->topIndex];

*stack*->arr[*stack*->topIndex] = NULL;

*stack*->topIndex--;

  return el;

}

// Возвращает индекс добавленного элемента в случае если его удалось добавить иначе выбрасывает ошибку

// Увеличивает значение topIndex на 1

*template* <*typename* T>

*int* push(Stack<T> \**stack*, T \**el*) {

  if (*stack*->topIndex >= (*int*)*stack*->size - 1) {

    throw STACK\_OVERFLOW;

  }

*stack*->arr[++*stack*->topIndex] = *el*;

  return *stack*->topIndex;

}

// Умножение строки на число n равносильно повторению строки n раз

string operator \*(const string &*str*, const size\_t &*times*) {

  string res = "";

  for (size\_t i = 0; i < *times*; i++) {

    res += *str*;

  }

  return res;

}

// Конкатенация стоки и символа

string operator +(const string &*str*, const *char* &*c*) {

  return *str* + string(1, *c*);

}

string operator +(const *char* &*c*, const string &*str*) {

  return string(1, *c*) + *str*;

}

// Обрезает строку до максимальной длины и выводит в консоль. Выравнивает по правому краю.

// Если строка больше максимальной длины то обрезает её

*void* outputFormatted(string *str*, size\_t *maxChars*) {

  cout.width(*maxChars*);

  if (*str*.length() > *maxChars*) {

*str* = *str*.substr(0, *maxChars* - 3) + "...";

  }

  cout << right << *str*;

}

*void* printTable(Stack<Book> \**stack*) {

  if (isEmpty(*stack*)) {

    cout << "Стопка книг пустая..." << endl;

    return;

  }

  const string TABLE\_PARTS = "—|";

  const string SPACE = " ";

  string blankLine = TABLE\_PARTS[1] + SPACE \* (1 + NUMBER\_WIDTH + 1) + TABLE\_PARTS[1] + SPACE \* (1 + TITLE\_WIDTH + 1)

    + TABLE\_PARTS[1] + SPACE \* (1 + NUMBER\_OF\_PAGES\_WIDTH + 1) + TABLE\_PARTS[1] + SPACE \* (1 + AUTHOR\_WIDTH + 1) + TABLE\_PARTS[1] + '\n';

  string dividingLine = string(1, TABLE\_PARTS[0]) \* (2 + NUMBER\_WIDTH + 3 + TITLE\_WIDTH + 3 + NUMBER\_OF\_PAGES\_WIDTH + 3 + AUTHOR\_WIDTH + 2) + '\n';

  cout << dividingLine << blankLine;

  cout << TABLE\_PARTS[1] << SPACE;

  outputFormatted(string("№"), NUMBER\_WIDTH);

  cout << SPACE << TABLE\_PARTS[1] << SPACE;

  outputFormatted(string("Название"), TITLE\_WIDTH);

  cout << SPACE << TABLE\_PARTS[1] << SPACE;

  outputFormatted(string("К-во стр"), NUMBER\_OF\_PAGES\_WIDTH);

  cout << SPACE << TABLE\_PARTS[1] << SPACE;

  outputFormatted(string("Автор"), AUTHOR\_WIDTH);

  cout << SPACE << TABLE\_PARTS[1] << endl;

  cout << blankLine << dividingLine;

  for (*int* i = getSize(*stack*) - 1; i >= 0; i--) {

    cout << blankLine;

    cout << TABLE\_PARTS[1] << SPACE;

    outputFormatted(to\_string(getSize(*stack*) - i), NUMBER\_WIDTH);

    cout << SPACE << TABLE\_PARTS[1] << SPACE;

    outputFormatted(*stack*->arr[i]->title, TITLE\_WIDTH);

    cout << SPACE << TABLE\_PARTS[1] << SPACE;

    outputFormatted(to\_string(*stack*->arr[i]->numberOfPages), NUMBER\_OF\_PAGES\_WIDTH);

    cout << SPACE << TABLE\_PARTS[1] << SPACE;

    outputFormatted(*stack*->arr[i]->author, AUTHOR\_WIDTH);

    cout << SPACE << TABLE\_PARTS[1] << endl;

    cout << blankLine << dividingLine;

  }

}

*void* clearInput() {

  cin.clear();

  while ((*char*)cin.get() != '\n');

}

*int* inputDigit() {

*char* c = cin.get();

  if ((*int*)c > 47 && (*int*)c < 58 && (*char*)cin.get() == '\n') {

    return c - '0';

  }

  clearInput();

  return -1;

}

*void* stackActionsLoop(Stack<Book> \**stack*) {

*bool* stop = false;

*int* currentAction = STACK\_ACTIONS;

  Book \*currentBook = NULL;

  while (!stop) {

    switch (currentAction) {

      case STACK\_ACTIONS:

        cout << "Введите цифру. Варианты действий:" << endl << "1. Вывести таблицу" << endl

          << "2. Взять книгу" << endl << "3. Добавить книгу" << endl << "4. Выход" << endl << endl;

        switch (inputDigit()) {

          // 1. Вывести таблицу

          case 1:

            cout << endl;

            printTable(*stack*);

            break;

          // 2. Взять книгу

          case 2:

            try {

              currentBook = pop(*stack*);

            } catch (Errors &e) {

              if (e == STACK\_UNDERFLOW){

                cout << endl << "Стопка книг пустая..." << endl;

              }

              break;

            }

            currentAction = BOOK\_ACTIONS;

            break;

          // 3. Добавить книгу

          case 3:

            currentAction = ADDING\_BOOK;

            break;

          // 4. Выход

          case 4:

            currentAction = EXIT;

            break;

        default:

          cout << endl << "Некорректный ввод" << endl;

        }

        break;

      case BOOK\_ACTIONS:

        cout << "В ваших руках книга. Варианты действий:" << endl << "1. Подробнее" << endl

          << "2. Изменить книгу" << endl << "3. Вернуть книгу" << endl << "4. Выбросить книгу" << endl << endl;

        switch (inputDigit()) {

          // 1. Подробнее

          case 1:

            cout << endl;

            printBook(currentBook);

            break;

          // 2. Изменить книгу

          case 2:

            cout << endl << "Введите цифру, которая отвечает за параметр что вы хотите изменить. Варианты:" << endl

              << "1. Название" << endl << "2. Количество страниц" << endl << "3. Автор" << endl << endl;

            switch (inputDigit()) {

              // 1. Изменить название

              case 1:

                cout << endl << "Введите новое название книги. Текущее: " << currentBook->title << endl;

                getline(cin, currentBook->title);

                cout << endl << "Вы изменили название книги.";

                break;

              // 3. Изменить автора

              case 3:

                cout << endl << "Введите имя нового автора книги. Текущее: " << currentBook->author << endl;

                getline(cin, currentBook->author);

                cout << endl << "Вы изменили автора книги.";

                break;

              // 2. Изменить количество страниц

              // 2 и 3 пункт в коде поменяны местами для того чтобы можно было вызывать кусок кода из default при некорректном вводе

              case 2: {

*int* originalNumberOfPages = currentBook->numberOfPages;

                cout << endl << "Введите новое количество страниц книги. Текущее: " << originalNumberOfPages << endl;

                cin >> currentBook->numberOfPages;

                clearInput();

                if (currentBook->numberOfPages > 0) {

                  cout << endl << "Вы изменили количество страниц";

                  break;

                }

                currentBook->numberOfPages = originalNumberOfPages;

              }

              default:

                cout << endl << "Некорректный ввод";

            }

            cout << endl;

            break;

          // 3. Вернуть книгу

          case 3:

            push(*stack*, currentBook);

            currentBook = NULL;

            currentAction = STACK\_ACTIONS;

            break;

          // 4. Выбросить книгу

          case 4:

            delete currentBook;

            currentBook = NULL;

            currentAction = STACK\_ACTIONS;

            break;

          default:

            cout << endl << "Некорректный ввод" << endl;

        }

        break;

      case ADDING\_BOOK: {

        string input;

        cout << "Введите название, количество страниц и автора в формате \"Н | КС | А\"" << endl;

        getline(cin, input);

        Book \*tmpBook;

        try {

          tmpBook = new Book(input);

          push(*stack*, tmpBook);

          cout << endl << "Книга добавлена" << endl;

        } catch (Errors &e) {

          switch (e) {

            case BAD\_FORMAT:

              cout << endl << "Некорректный ввод" << endl;

              break;

            case STACK\_OVERFLOW:

              delete tmpBook;

              cout << endl << "Стопка переполнена" << endl;

              break;

          }

        }

        currentAction = STACK\_ACTIONS;

        break;

      }

      case EXIT:

        stop = true;

        break;

      default:

        cout << "Ух ты..." << endl;

        stop = true;

    }

    cout << endl;

  }

}

*int* main() {

  system("chcp 1251");

  size\_t size;

*bool* cond;

  Stack<Book> \*stack;

  do {

    cond = false;

    try {

      cout << "Введите размер стопки книг: ";

      cin >> size;

      clearInput();

      if (!size) throw bad\_array\_new\_length();

      stack = new Stack<Book>(size);

    } catch (bad\_array\_new\_length) {

      cond = true;

      cout << "Некорректный размер стопки" << endl;

    }

    cout << endl;

  } while (cond);

  stackActionsLoop(stack);

  delete stack;

  return 0;

}

Завдання №2

Структура даних Товар має наступні поля: назва, ціна, строк придатності

(днів). Виконати наступні завдання:

1) Створити стрічку (чергу) товарів на касі магазину, розмір стрічки 5.

2) Покласти на неї 4 різні товари.

3) Взяти перший товар, застосувати до нього купон на знижку 25% та

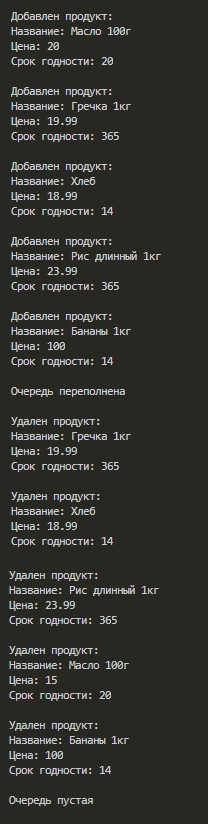
повернути на стрічку.

4) Покласти ще 3 товари.

5) Забрати 3 товари зі стрічки.

6) Забрати ще 4 товари зі стрічки.

Результат



Лістинг

#include <iostream>

#include <clocale>

#include <string>

#define SIZE\_OF\_QUEUE 5

using *namespace* std;

*enum* QueueErrors {

  QUEUE\_UNDERFLOW,

  QUEUE\_OVERFLOW,

};

*struct* Product {

  string name;

*float* price;

*int* expirationDate;

};

ostream& operator<< (ostream &*out*, const Product &*product*) {

*out* << "Название: " << *product*.name << endl << "Цена: " << *product*.price

    << endl << "Срок годности: " << *product*.expirationDate;

  return *out*;

}

*template* <*typename* T>

*struct* Queue {

  T arr[SIZE\_OF\_QUEUE];

*int* size = SIZE\_OF\_QUEUE;

*int* head = -1;

*int* tail = -1;

};

*template* <*typename* T>

*bool* isEmpty(Queue<T> \**queue*) {

  return *queue*->head == -1;

}

*template* <*typename* T>

*int* getSize(Queue<T> \**queue*) {

  if (isEmpty(*queue*)) return 0;

  if (*queue*->tail < *queue*->head) {

    return *queue*->size - *queue*->head + *queue*->tail + 1;

  }

  return *queue*->tail - *queue*->head + 1;

}

*template* <*typename* T>

T front(Queue<T> \**queue*) {

  if (isEmpty(*queue*)) {

    throw QUEUE\_UNDERFLOW;

  }

  return *queue*->arr[*queue*->head];

}

*template* <*typename* T>

*void* push(Queue<T> \**queue*, T *el*) {

  if ((*queue*->tail + 1) % *queue*->size == *queue*->head) {

    throw QUEUE\_OVERFLOW;

  }

*queue*->tail = (*queue*->tail + 1) % *queue*->size;

*queue*->arr[*queue*->tail] = *el*;

  if (isEmpty(*queue*)) *queue*->head++;

}

*template* <*typename* T>

T pop(Queue<T> \**queue*) {

  if (isEmpty(*queue*)) {

    throw QUEUE\_UNDERFLOW;

  }

  if (*queue*->head == *queue*->tail) {

    T tmp = *queue*->arr[*queue*->head];

*queue*->head = -1;

*queue*->tail = -1;

    return tmp;

  }

  T tmp = *queue*->arr[*queue*->head];

*queue*->head = (*queue*->head + 1) % *queue*->size;

  return tmp;

}

*template* <*typename* T>

T last(Queue<T> \**queue*) {

  if (isEmpty(*queue*)) {

    throw QUEUE\_UNDERFLOW;

  }

  return *queue*->arr[*queue*->tail];

}

*int* main() {

  setlocale(LC\_ALL, "rus");

  Queue<Product> tape;

  try {

    push(&tape, Product {"Масло 100г", 20, 20});

    cout << "Добавлен продукт:" << endl << last(&tape) << endl << endl;

    push(&tape, Product {"Гречка 1кг", 19.99, 365});

    cout << "Добавлен продукт:" << endl << last(&tape) << endl << endl;

    push(&tape, Product {"Хлеб", 18.99, 14});

    cout << "Добавлен продукт:" << endl << last(&tape) << endl << endl;

    push(&tape, Product {"Рис длинный 1кг", 23.99, 365});

    cout << "Добавлен продукт:" << endl << last(&tape) << endl << endl;

  } catch (QueueErrors e) {

    if (e != QUEUE\_OVERFLOW) throw e;

    cout << "Очередь переполнена" << endl << endl;

  }

  Product tmp = pop(&tape);

  tmp.price \*= .75;

  push(&tape, tmp);

  try {

    push(&tape, Product {"Бананы 1кг", 100, 14});

    cout << "Добавлен продукт:" << endl << last(&tape) << endl << endl;

    push(&tape, Product {"Апельсины 10кг", 600, 14});

    cout << "Добавлен продукт:" << endl << last(&tape) << endl << endl;

    push(&tape, Product {"Лимон 100г", 14, 14});

    cout << "Добавлен продукт:" << endl << last(&tape) << endl << endl;

  } catch (QueueErrors e) {

    if (e != QUEUE\_OVERFLOW) throw e;

    cout << "Очередь переполнена" << endl << endl;

  }

  try {

    for (*int* i = 0; i < 3; i++) {

      cout << "Удален продукт:" << endl << pop(&tape) << endl << endl;

    }

  } catch (QueueErrors e) {

    if (e != QUEUE\_UNDERFLOW) throw e;

    cout << "Очередь пустая" << endl << endl;

  }

  try {

    for (*int* i = 0; i < 4; i++) {

      cout << "Удален продукт:" << endl << pop(&tape) << endl << endl;

    }

  } catch (QueueErrors e) {

    if (e != QUEUE\_UNDERFLOW) throw e;

    cout << "Очередь пустая" << endl << endl;

  }

  return 0;

}

Висновок

Я ознайомився та дослідив принципи FIFO, LIFO та лінійні структури даних: стек та чергу. Реалізував їх на основі «статичних» масивів, набув навичок роботи зі структурами та вказівниками на мові програмування С/C++.