МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра защиты информации

**

**ОТЧЁТ**

**по лабораторной работе №2**

**«**Алгоритмы и абстрактные структуры данных**»**

**по дисциплине: «***Программирование***»**

Выполнил:Проверил:

Студент гр. «АБс-222», «АВТФ» *доцент кафедры ЗИ*

*Азява Д.А. Архипова А. Б.*

«\_\_\_» \_\_\_\_\_\_ 2023 г.«\_\_\_» \_\_\_\_\_\_ 2023 г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (подпись)

Новосибирск 2023

Цели и задачи работы: изучение алгоритмов работы с абстрактными структурами данных.

**Задание 1:** На вход передаётся строка, в которой содержатся различные символы, включая символы скобок "({[". Необходимо реализовать алгоритм, который определяет, правильно ли вложены скобки, используя стек. Например, последовательность "({})" - верная, последовательности "({)}" и "(][)" - неверные.

Таблица 1 - Листинг программы

|  |  |
| --- | --- |
| Файл | Код |
| Task-1.c | #include "stack.h"  #include <stdio.h>  #include <string.h>  bool isRight(char\* text)  {  int size = strlen(text);  for (int i = 0; i < size; i++)  {  switch (text[i])  {  case '(':  push\_stack(text[i]);  break;  case ')':  if (stackIsEptpy() || pop\_stack() != '(')  {  return false;  }  break;  case '[':  push\_stack(text[i]);  break;  case ']':  if (stackIsEptpy() || pop\_stack() != '[')  {  return false;  }  break;  case '{':  push\_stack(text[i]);  break;  case '}':  if (stackIsEptpy() || pop\_stack() != '{')  {  return false;  }  break;  default:  break;  }  }  return stackIsEptpy();  }  int main()  {  char\* test1 = "(1te{st)}";  char\* test2 = "(te[s]t)";  printf("Test 2: %s - ", test2);  if (isRight(test2))  {  printf("right!\n");  }  else  {  printf("incorrect!\n");  }  } |

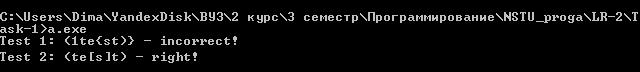


Рисунок 1 – Результат работы программы

**Задание 2:** Необходимо реализовать алгоритм, который должен разбить множество натуральных чисел на непересекающиеся подмножества, все из которых имеют равную сумму. Если множество не может быть поделено таким образом, вывести сообщение об ошибке.

Таблица 2 - Листинг программы

|  |  |
| --- | --- |
| Файл | Код |
| Task-2.c | #include <stdio.h>  #include <stdbool.h>  #define size 12  #define kol 5  bool checkSum(int sumLeft[], int k)  {  int res = true;  for (int i = 0; i < k; i++)  {  if (sumLeft[i] != 0)  {  res = false;  }  }  return res;  }  bool subsetSum(int S[], int n, int sumLeft[], int A[], int k)  {  if (checkSum(sumLeft, k))  {  return true;  }  if (n < 0)  {  return false;  }  bool result = false;  for (int i = 0; i < k; i++)  {  if (!result && (sumLeft[i] - S[n]) >= 0)  {  A[n] = i + 1;  sumLeft[i] = sumLeft[i] - S[n];  result = subsetSum(S, n - 1, sumLeft, A, k);  sumLeft[i] = sumLeft[i] + S[n];  }  }  return result;  }  int accumulate(int S[])  {  int res = 0;  for (int i = 0; i < size; i++)  {  res += S[i];  }  return res;  }  void partition(int S[], int n, int k)  {  if (n < k)  {  printf("k-partition of set S is not possible\n");  return;  }  int sum = accumulate(S);  int A[size], sumLeft[kol];  for (int i = 0; i < k; i++)  {  sumLeft[i] = sum / k;  }  bool result = !(sum % k) && subsetSum(S, n - 1, sumLeft, A, k);  if (!result)  {  printf("k-partition of set S is not possible\n");  return;  }  for (int i = 0; i < k; i++)  {  printf("Partition %d is ", i);  for (int j = 0; j < n; j++)  {  if (A[j] == i + 1)  {  printf("%d ", S[j]);  }  }  printf("\n");  }  }  int main()  {  int S[] = { 7, 3, 5, 12, 2, 1, 5, 3, 8, 4, 6, 4 };  printf("\nOutput: ");  for (int i = 0; i < size; i++)  {  printf("%d ", S[i]);  }  printf("\n\n");  partition(S, size, kol);  return 0;  } |

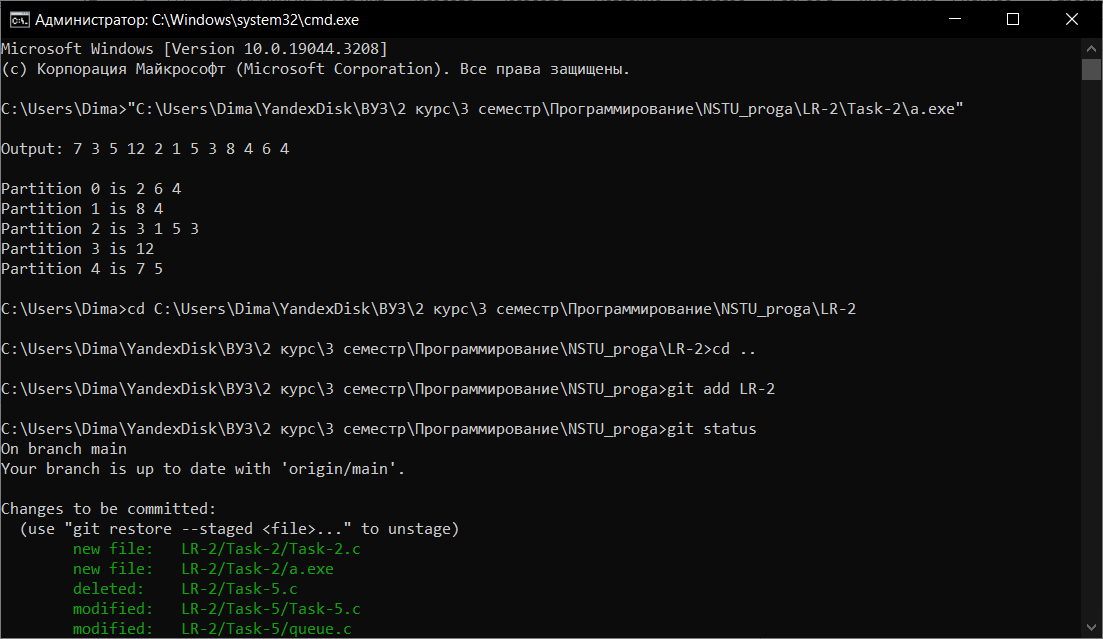


Рисунок 2 – Результат работы программы

**Задание 3:** Необходимо реализовать алгоритм, который находит в круговом массиве подмассив с наибольшей суммой.

Таблица 3 - Листинг программы

|  |  |
| --- | --- |
| Файл | Код |
| Task-3.c | #include <stdio.h>  #include <stdlib.h>  typedef struct need {  int startIndex;  int endIndex;  int maxSum;  }need;  void letsDoIt(int arr[], need\* myStruct, int size)  {  int currSum = 0;  myStruct->startIndex = 0;  myStruct->maxSum = 0;  for (myStruct->endIndex = 0; myStruct->endIndex < size; myStruct->endIndex++)  {  currSum += arr[myStruct->endIndex];    if (currSum > myStruct->maxSum)  {  myStruct->maxSum = currSum;  }  else if (currSum < 0)  {  currSum = 0;  myStruct->startIndex = myStruct->endIndex + 1;  }  }  myStruct->endIndex--;  }  int main()  {  need\* myStruct = (need\*)malloc(sizeof(need));  int arr[10] = { 4, -7, 1, 5, -4, 0, -3, 2, 4, 1 };  int size = sizeof(arr) / sizeof(int);  //int arr[5] = { 4, -7, 5, -1, 8 };  printf("\nInput: ");  for (int i = 0; i < size-1; i++)  {  printf("%d ", arr[i]);  }  letsDoIt(arr, myStruct, size);  printf("\nOutput: ");  for (int i = myStruct->startIndex; i <= myStruct->endIndex; i++)  {  printf("%d ", arr[i]);  }  printf("\tsum: %d\n", myStruct->maxSum);  } |

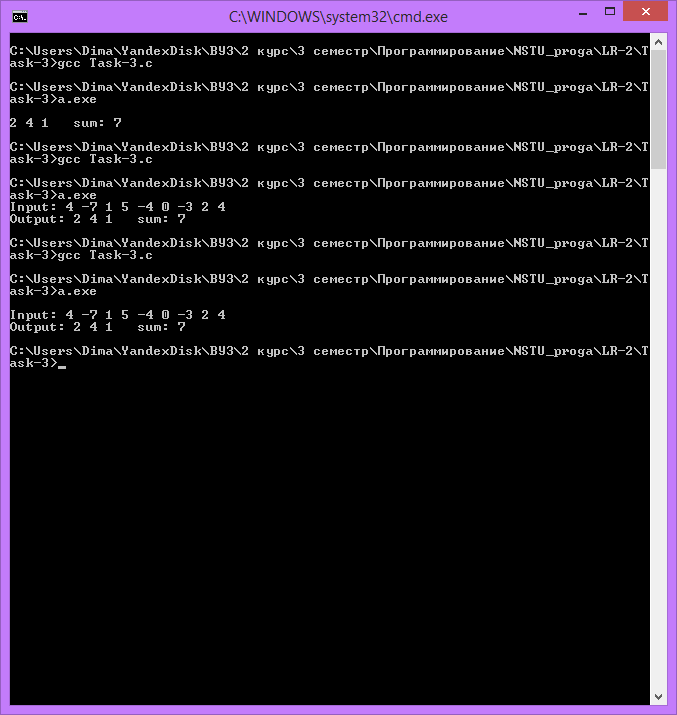


Рисунок 3 - Результат работы программы

**Задание 4:** Необходимо реализовать алгоритм, который будет проверять, является ли дерево binary search tree(BST).

Дерево не может быть BST, если левое поддерево содержит любое значение больше, чем значение узла или правое поддерево содержит любое значение меньше, чем значение узла. Другими словами, значение узла должно быть больше максимального в левом поддереве и меньше минимального в правом поддереве. Таким образом, алгоритм рекурсивно передвигается по дереву, и сравнивает минимальное и максимальное значения с узлом.

В таблице 4 представлена функция, выполняющая проверку.

Таблица 4 - Листинг программы

|  |  |
| --- | --- |
| Файл | Код |
| tree.c | #include "tree.h"  #include <stdio.h>  #include <stdlib.h>  #incluse <stdbool.h>  bool isBST(node\* root)  {  if (root == NULL)  {  return true;  }  int leftMax = 0;  int rightMax = 0;  if (root->left != NULL)  {  leftMax = bstMax(root->left);  }  else if (root->right != NULL)  {  if (bstMin(root->right) < root->key)  {  return false;  }  }  if (root->right != NULL)  {  rightMax = bstMin(root->right);  }  else if (root->left != NULL)  {  if (bstMax(root->left) > root->key)  {  return false;  }  }  bool left = isBST(root->left);  bool right = isBST(root->right);  if (left == true && right == true)  {  return true;  }  else  {  return false;  }  }  int bstMax(node\* root)  {  while (root->right != NULL)  {  root = root->right;  }  return root->key;  }  int bstMin(node\* root)  {  while (root->left != NULL)  {  root = root->left;  }  return root->key;  } |

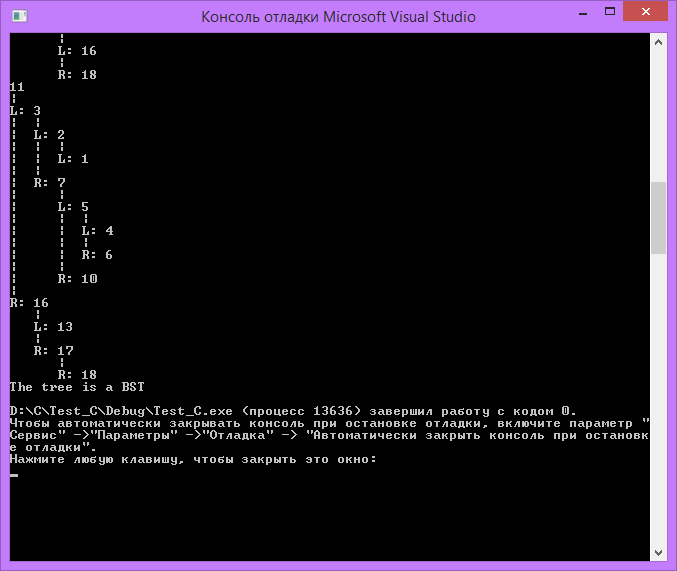


Рисунок 4 - Результат работы программы

**Задание 5:** Дана шахматная доска размером NxN и позиция коня. Задается новая позиция. Необходимо вычислить и вывести кратчайший путь до этой позиции, используя очередь.

Таблица 5 - Листинг программы

|  |  |
| --- | --- |
| Файл | Код |
| Task-5.c | #include "queue.h"  #include <stdio.h>  #include <string.h>  #include <stdlib.h>  #define M 4  int init\_position(int x, int y)  {  x \*= 10;  return x + y;  }  bool OnBoard(int k, int s)  {  if (k >= 1 && k <= M && s >= 1 && s <= M)  {  return true;  }  else  {  return false;  }  }  void letEatBee(int x, int y, int X, int Y)  {  int dx[8] = { -2, -1, 1, 2, -2, -1, 1, 2 };  int dy[8] = { -1, -2, -2, -1, 1, 2, 2, 1 };  push\_queue(init\_position(x, y));    printf("\t%d\n", init\_position(x, y));  bool visited[M + 1][M + 1] = { false };  visited[x][y] = true;  int kMove[M + 1][M + 1] = { 0 };  while (!is\_empty())  {  int z = pop\_queue();  int x\_z = z / 10;  int y\_z = z % 10;  if (x\_z == X && y\_z == Y)  {  printf("Minimum number of moves: %d\n", kMove[x\_z][y\_z]);  return;  }  for (int i = 0; i < 8; i++)  {  int x\_new = x\_z + dx[i];  int y\_new = y\_z + dy[i];  if (OnBoard(x\_new, y\_new) && !visited[x\_new][y\_new])  {  visited[x\_new][y\_new] = true;  kMove[x\_new][y\_new] = kMove[x\_z][y\_z] + 1;  push\_queue(init\_position(x\_new, y\_new));  printf("\t%d\n", init\_position(x\_new, y\_new));  }  }  }  printf("ERROR: it's impossible\n");  }  int main()  {  int x = 1, y = 4;  letEatBee(x, y, 3, 1);  } |

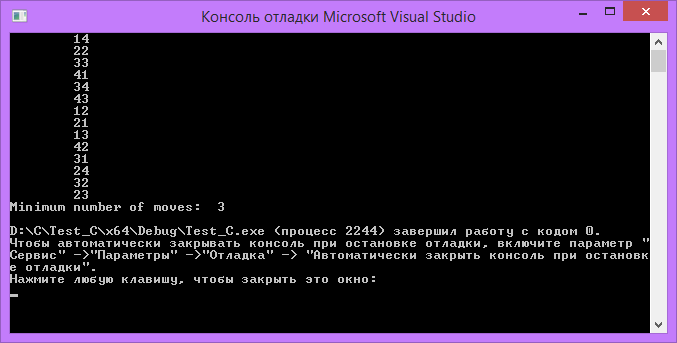


Рисунок 5 - Результат работ программы

**Задание 6:** Необходимо реализовать алгоритм, который определяет, являются ли строки изоморфными. Две строки a и b считаются изоморфными, если символы в a можно заменить на b. Символ сопоставляется только одному другому, порядок символов должен сохраняться.

Таблица 6 - Листинг программы

|  |  |
| --- | --- |
| Файл | Код |
| Task-6.c | #include "hash\_table.h"  #include <stdio.h>  #include <string.h>  #include <stdbool.h>  char\* append(char\* s, char c)  {  s[0] = c;  s[1] = '\0';  return s;  }  bool isIsomorphic(char\* first, char\* second)  {  int size1 = strlen(first);  int size2 = strlen(second);  if (size1 != size2)  {  return false;  }  HT\* table = create\_table(size2);  char fst[2] = " ", snd[2] = " ";  strcpy(fst, append(fst, first[0]));  strcpy(snd, append(snd, second[0]));  ht\_insert(table, fst, snd);  for (int i = 1; i < strlen(first); i++)  {  strcpy(fst, append(fst, first[i]));  strcpy(snd, append(snd, second[i]));  if (isKey(table, fst))  {  if ( strcmp(ht\_search(table, fst), snd) != 0)  {  return false;  }  }  else  {  ht\_insert(table, fst, snd);  }  }  return true;  }  int main()  {  char\* first = "ABCA";  char\* second = "XYZX";  if (isIsomorphic(first, second))  {  printf("%s and %s are Isomorphic\n", first, second);  }  else  {  printf("%s and %s are NOT Isomorphic\n", first, second);  }  } |



Рисунок 6 - Результат работы программы

Вывод: В ходе выполнения лабораторной работы были реализованы алгоритмы работы с абстрактными структурами данных, было рассмотрено практическое применение абстрактных структур данных и улучшена их реализация.