**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра АПУ**

**ОТЧЕТ**

**по лабораторной работе №9**

**по дисциплине «Программирование»**

| Студент гр. 3391 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Николаев В.Ю. |
| --- | --- | --- |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Писарев А.С. |

Санкт-Петербург

2024 г.

# Исходная формулировка

Разработать программу построения и использования структуры быстрого поиска чисел с первичным разделением интервала искомых чисел на базе хэш-массива фиксированной длины. С каждым элементом хэш-массива должен связываться корень бинарного дерева поиска, в которым должны размещаться ключи из соответствующего поддиапазона.

# Ход работы программы

Программа начинается с определения структуры WordInfo, которая содержит бинарное дерево поиска.

| 6  7  8  9 | struct NumberInfo  {  BinarySearchTree tree;  }; |
| --- | --- |

Функция hashFunction принимает число и возвращает индекс в хеш-таблице.

| 11  12  13  14 | int hashFunction(const int& number)  {  return (number % MAX\_SIZE + MAX\_SIZE) % MAX\_SIZE;  } |
| --- | --- |

Функция insertNumber вставляет число в соответствующее дерево в хеш-таблице.

| 16  17  18  19  20  21 | void insertNumber(NumberInfo\* hashTable, const int& number)  {  int index = hashFunction(number);    hashTable[index].tree.insert(number);  } |
| --- | --- |

Функция searchNumber принимает число, вычисляет его хеш и ищет его в соответствующей ячейке в хеш-таблице. Если число найдено, функция возвращает true, в противном случае - false.

| 23  24  25  26  27  28  29  30  31 | bool searchNumber(NumberInfo\* hashTable, const int& number)  {  int index = hashFunction(number);    if (hashTable[index].tree.find(number))  return true;  else  return false;  } |
| --- | --- |

В main функции создается хеш-таблица и предоставляется меню для вставки чисел, поиска чисел и выхода из программы

| 33  34  35  40  41    49  50  51  55  56  57  64  65  66  68  69  70  71  72  73  76  77  78  79  80 | int main()  {  NumberInfo\* hashTable = new NumberInfo[MAX\_SIZE];  // ...  do  {  // ...  switch (choice)  {  case 1:  // Вставка числа  break;  case 2:  // Поиск числа  break;  case 3:  // Выход из программы  break;  default:  std::cout << "Invalid choice\n";  break;  }  // ...  }  while (choice != 3);    return 0;  } |
| --- | --- |

# 

# Текст программы

| Main.cpp | |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80 | #include "BinarySearchTree.h"  #include <iostream>  const int MAX\_SIZE = 2;  struct NumberInfo  {  BinarySearchTree tree;  };  int hashFunction(const int& number)  {  return (number % MAX\_SIZE + MAX\_SIZE) % MAX\_SIZE;  }  void insertNumber(NumberInfo\* hashTable, const int& number)  {  int index = hashFunction(number);    hashTable[index].tree.insert(number);  }  bool searchNumber(NumberInfo\* hashTable, const int& number)  {  int index = hashFunction(number);    if (hashTable[index].tree.find(number))  return true;  else  return false;  }  int main()  {  NumberInfo\* hashTable = new NumberInfo[MAX\_SIZE];    int choice;  int number;    do  {  std::cout << "Menu:\n";  std::cout << "1. Insert number\n";  std::cout << "2. Search number\n";  std::cout << "3. Exit\n";  std::cout << "Enter your choice: ";  std::cin >> choice;    switch (choice)  {  case 1:  std::cout << "Enter number to insert: ";  std::cin >> number;  insertNumber(hashTable, number);  break;  case 2:  std::cout << "Enter number to search: ";  std::cin >> number;  if (searchNumber(hashTable, number))  std::cout << "Number found\n";  else  std::cout << "Number not found\n";  break;  case 3:  std::cout << "Exiting...\n";  break;  default:  std::cout << "Invalid choice\n";  break;  }    std::cout << std::endl;  }  while (choice != 3);    return 0;  } |
| BinarySearchTree.h | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147 | #pragma once  #include <iostream>  class BinarySearchTree  {  private:  struct Node  {  int data;  Node\* left;  Node\* right;  Node(int data, Node\* left, Node\* right):  data(data),  left(left),  right(right)  {}  ~Node()  {  delete left;  delete right;  }  };  Node\* root;  void insert(Node\*& node, const int& data)  {  if (node == nullptr)  {  node = new Node(data, nullptr, nullptr);  }  else if (data < node->data)  insert(node->left, data);  else  insert(node->right, data);    }  bool find(Node\* node, const int& data) const  {  if (node == nullptr)  return false;  else if (data == node->data)  return true;  else if (data < node->data)  return find(node->left, data);  else  return find(node->right, data);  }  void printHelper(int level, Node\* node, int mode, char\* modes) const  {  if (node == nullptr)  return;  modes[level + 1] = 'r';  printHelper(level + 1, node->right, 1, modes);  modes[level] = (mode == 0) ? 'm' : (mode == 1) ? 'v' : 'l';  for (int i = 0; i < level; i++)  if (modes[i] == 'v')  std::cout << "│ ";  else  std::cout << " ";  switch (mode)  {  case 1:  std::cout << "┌─";  break;  case 0:  std::cout << "├─";  break;    case -1:  std::cout << "└─";  break;  }  std::cout << node->data << std::endl;  modes[level + 1] = 'v';  printHelper(level + 1, node->left, -1, modes);  }  int height(Node\* node) const  {  if (node == nullptr)  return 0;  else  {  int leftHeight = height(node->left);  int rightHeight = height(node->right);  return 1 + ((leftHeight > rightHeight) ? leftHeight : rightHeight);  }  }  public:  BinarySearchTree()  {  root = nullptr;  }  void insert(const int& data)  {  insert(root, data);  }  bool find(const int& data) const  {  return find(root, data);  }  bool empty() const  {  return root == nullptr;  }  void print() const  {  int h = height(root);  char \*modes = new char[h + 1];  modes[0] = 'm';  for (int i = 1; i < h + 1; i++)  modes[i] = 'v';  printHelper(0, root, 0, modes);  delete[] modes;  }  void clear()  {  delete root;  root = nullptr;  }  ~BinarySearchTree()  {  delete root;  }  }; |

# Результаты работы программы

|  |  |
| --- | --- |