

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННО БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Учебный Центр Информационных Технологий «Информатика»



Лабораторная работа № 4
по дисциплине «Объектно-ориентированное программирование»

Направление подготовки: 230105 - «Программное обеспечение вычислительной техники и автоматизированных систем»

Выполнил слушатель: Бройтман Е.Д.
Вариант: №5
Дата сдачи:
Преподаватель: Силов Я.В.

Новосибирск, 2016г.

1. Задание

Необходимо разработать шаблон класса, реализующий структуру данных (контейнер). Параметры шаблона передаются через угловые скобки, и позволяют строить контейнер на статическом массиве.

Шаблон, помимо структуры хранения, должен включать в себя:

- 1) Функции добавления значения (оператор индексирования для массивов, Push для стека, Into, Add и т.п.).
- 2) Функции получения значения.
- 3) Дополнительные функции (проверка количества элементов, переполнения и т.п.).

Все функции должны проверять корректность входных значений, проверять переполнение (в общем, обеспечивать полностью корректную работу контейнера).

В рамках данной работы, рекомендуется хранить непосредственно объекты (а не указатели на них).

Пользовательский интерфейс должен продемонстрировать работу шаблона.

В простейшем случае, это занесение ряда значений, а потом их получение с выводом на экран.

Демонстрацию работы шаблона можно провести на любом простом или сложном типе данных.

ОБЯЗАТЕЛЬНО продемонстрировать работу механизма обработки переполнений.

Тип контейнера - двоичное дерево.

2. Структурное описание

В данной лабораторной работе создается шаблонный класс двоичного дерева **BSTree** с вложенным в него классом **Node**.

Для объявления интерфейса, реализации методов классов и полей классов **BSTree** и **Node**, объявления прототипа функции void Menu() и глобальных констант использован заголовочный файл «lab4.h».

В файле "Menu.cpp" реализовано пользовательское меню, обеспечивающее:

- Создание нового объекта контейнера-дерева для целых чисел на 10 вершин.
- Ввод новых значений для объекта с заданным номером.
- Добавление вершин дерева.
- Просмотр дерева.

В файле "Main.cpp" реализованы вызовы функций обеспечения работы с кириллической кодировкой и функции меню.

Поля и методы шаблонного класса **BSTree**:

- <class Type> - параметр шаблона класса.
- Node *Root - указатель на корень дерева (private).
- int Count - количество вершин дерева (private).
- BSTree() - конструктор по умолчанию (public).
- ~BSTree() - деструктор (public).
- Node* ClearAll(Node*) - удаление вершин дерева (public).
- Node* Paste(Node *, const Type &) - вставка вершины дерева (public).
- int GetCount() - получение количества вершин дерева (public).

- Node *GetRoot() - получение указателя на корень дерева (public).
- int TreeHeight(Node * LocalRoot) - вычисление высоты дерева с учетом веток к нулевым вершинам (public).
- void Show(Node * LocalRoot) - вывод дерева на экран (public).
- void ShowLevel(Node * LocalRoot, int ReqLevel, int Height, int TempLevel) - вывод одного уровня дерева на экран (public).

Поля и методы класса **Node**, вложенного в шаблонный класс **BSTree**:

- Type Data - данные вершины (private).
- Node *Left - указатель на вершину левого поддерева (private).
- Node *Right - указатель на вершину правого поддерева (private).
- Node(const Type &) - конструктор с параметром (public).
- Node(const Node &) - конструктор копирования (public).
- Type GetData() - получение данных в текущей вершине дерева (public).
- void SetData(const Type &) - запись данных в текущую вершину дерева (public).
- Node * GetLeft() - получение указателя на вершину левого поддерева текущей вершины дерева (public).
- void SetLeft(Node *) - запись указателя на вершину левого поддерева в текущую вершину дерева (public).
- Node * GetRight() - получение указателя на вершину правого поддерева текущей вершины дерева (public).
- void SetRight(Node *) - запись указателя на вершину правого поддерева в текущую вершину дерева (public).

3. Функциональное описание

Рассмотрим реализацию методов шаблонного класса **BSTree**.

1. Конструктор класса по умолчанию BSTree() – запускается автоматически при создании нового объекта класса без параметров, инициализирует закрытые поля класса значениями по умолчанию. Инициализация выполнена списком инициализации, значения по умолчанию Root=NULL, Count=0.

2. Деструктор ~BSTree(). Вызывается при удалении экземпляра объекта дерева. Освобождает память из-под вершин дерева, вызывая метод Node* ClearAll(Node*). Выводит сообщение об оставшемся количестве вершин дерева.

3. Удаление вершин дерева Node* ClearAll(Node*). Выполняет рекурсивный обход дерева, удаляя концевые вершины (вершины у которых указатели на оба потомка == NULL). Вершина, у которой только что удалены оба потомка, становится концевой и тоже удаляется.

4. Вставка вершины дерева Node * Paste(Node *, const Type &). При добавлении первой вершины дерева:

- Создается новая вершина дерева при помощи конструктора с параметром.
- Указателю на корень дерева присваивается адрес этой вершины.
- Счетчику Count присваивается 1.
- Возвращается значение указателя на корень дерева.

Иначе (дерево не пустое):

- Если указатель на текущую вершину ==NULL (вершина пустая):

- Count увеличивается на 1.
- Создается новая вершина дерева при помощи конструктора с параметром.
- Возвращается адрес этой вершины.
- Если текущая вершина не пустая:
 - Рекурсивно ищем подходящую свободную вершину:
 - Если значение данных в вершине больше вставляемого значения – ищем пустую вершину в левой ветке.
 - Иначе – в правой ветке.
 - Возвращается значение указателя на текущую вершину.

5. Получение количества вершин дерева `int GetCount()`. Возвращается значение количества вершин дерева – закрытый член класса.

6. Получение указателя на корень дерева `Node *GetRoot()`. Возвращается значение указателя на корень дерева – закрытый член класса.

7. Вычисление высоты дерева с учетом веток к нулевым вершинам `int TreeHeight(Node * LocalRoot)`. Начинаем с корневой вершины дерева.

Если вершина дерева пустая – дошли до конца ветви, возвращаем 0.

Для каждого потомка по порядку, принимая его за корень, повторяем действия, начиная с предыдущего пункта, при этом присваивая соответствующим максимальным длинам ветвей дерева для текущего корня величину, возвращенную в результате этих действий.

Возвращаем максимальное значение из вычисленных на предыдущих шагах максимальных длин ветвей каждого потомка дерева для текущего корня, увеличенное на 1.

8. Вывод дерева на экран `void Show(Node * LocalRoot) - (public)`. Вычисляем высоту дерева.

Выводим по порядку с новой строки каждый уровень дерева, начиная с нулевого.

9. Вывод одного уровня дерева на экран `void ShowLevel(Node * LocalRoot, int ReqLevel, int Height, int TempLevel)`.

9.1. Начинаем с корневой вершины дерева, нулевого уровня.

9.2. Задаем форматную строку для вывода одного значения вершины дерева. Форматная строка имеет вид «`%x<x>s`», где `x<x>` - одно- или двухразрядное десятичное целое число знакомест, занимаемых одним значением вершины дерева.

9.3. Инициализируем количество пробелов, которые нужно напечатать перед вершиной текущего уровня по формуле: количество знакомест для разделителя между вершинами нижнего уровня / 2. Здесь и далее все действия производятся в целочисленной арифметике.

9.4. Инициализируем количество пробелов, которые нужно напечатать после вершины текущего уровня по формуле: количество знакомест для разделителя между вершинами нижнего уровня минус количество пробелов, которые нужно напечатать перед вершиной текущего уровня.

9.5. Если текущий уровень меньше запрашиваемого:

9.5.1. Если вершина дерева не пустая – не дошли до конца ветви - для каждого потомка по порядку, принимая его за корень, и увеличивая номер текущего уровня на 1, повторяем действия, начиная с п. 7.2, таким образом, переходя к следующему уровню.

9.5.2. Иначе (вершина дерева пустая – дошли до конца ветви) - необходимо напечатать пробелы в запрашиваемом уровне (текстовый курсор на-

ходится в запрашиваемом уровне), чтобы восполнить отсутствующие элементы из-за нулевых вершин в текущем уровне и сохранить симметрию дерева:

9.5.2.1. Если количество потомков четное:

9.5.2.1.1. Повторяем количество раз равное разнице между высотой дерева минус 1 и номером текущего уровня:

9.5.2.1.1.1. Временной переменной 1 присваиваем значение количества пробелов, которые нужно напечатать перед вершиной текущего уровня.

9.5.2.1.1.2. Временной переменной 2 присваиваем значение количества пробелов, которые нужно напечатать после вершины текущего уровня.

9.5.2.1.1.3. Вычисляем количество пробелов, которые нужно напечатать перед вершиной по формуле: количество потомков вершины * (количество знакомест, занимаемых одной вершиной дерева + временная переменная 1 + временная переменная 2) / 2 - количество знакомест, занимаемых одной вершиной дерева / 2.

9.5.2.1.1.4. Вычисляем количество пробелов, которые нужно напечатать после вершины по формуле: количество пробелов, которые нужно напечатать перед вершиной + количество знакомест, занимаемых одной вершиной дерева / 2 - (количество знакомест, занимаемых одной вершиной дерева - количество знакомест, занимаемых одной вершиной дерева / 2).

9.5.2.2. Иначе (если количество потомков нечетное):

9.5.2.1.2. Повторяем количество раз равное разнице между высотой дерева минус 1 и номером текущего уровня:

9.5.2.1.2.1. Временной переменной 1 присваиваем значение количества пробелов, которые нужно напечатать перед вершиной текущего уровня.

9.5.2.1.2.2. Временной переменной 2 присваиваем значение количества пробелов, которые нужно напечатать после вершины текущего уровня.

9.5.2.1.2.3. Вычисляем количество пробелов, которые нужно напечатать перед вершиной по формуле: (количество потомков вершины - 1) * (количество знакомест, занимаемых одной вершиной дерева + временная переменная 1

+ временная переменная 2) / 2 + временная переменная 1.

9.5.2.1.2.4. Вычисляем количество пробелов, которые нужно напечатать после вершины по формуле: количество пробелов, которые нужно напечатать перед вершиной - временная переменная 1 + временная переменная 2.

9.5.2.3. Для сохранения симметрии дерева - вывод вычисленного количества пробелов перед вершиной, пробелов на месте вершины и вычисленного количества пробелов после вершины.

9.6. Иначе (находимся на запрашиваемом уровне):

9.6.1. Если количество потомков четное:

9.6.1.1. Повторяем количество раз равное разнице между высотой дерева минус 1 и номером текущего уровня:

9.6.1.1.1. Временной переменной 1 присваиваем значение количества пробелов, которые нужно напечатать перед вершиной текущего уровня.

9.6.1.1.2. Временной переменной 2 присваиваем значение количества пробелов, которые нужно напечатать после вершины текущего уровня.

9.6.1.1.3. Вычисляем количество пробелов, которые нужно напечатать перед вершиной по формуле: количество потомков вершины * (количество знакомест, занимаемых одной вершиной дерева + временная переменная 1 + временная переменная 2) / 2 - количество знакомест, занимаемых одной вершиной дерева / 2.

9.6.1.1.4. Вычисляем количество пробелов, которые нужно напечатать после вершины по формуле: количество пробелов, которые нужно напечатать перед вершиной + количество знакомест, занимаемых одной вершиной дерева / 2 - (количество знакомест, занимаемых одной вершиной дерева - количество знакомест, занимаемых одной вершиной дерева / 2).

9.6.2. Иначе (если количество потомков нечетное):

9.6.2.1. Повторяем количество раз равное разнице между высотой дерева минус 1 и номером текущего уровня:

9.6.2.1.1. Временной переменной 1 присваиваем значение количества пробелов, которые нужно напечатать перед вершиной текущего уровня.

9.6.2.1.2. Временной переменной 2 присваиваем значение количества пробелов, которые нужно напечатать после вершины текущего уровня.

9.6.2.1.3. Вычисляем количество пробелов, которые нужно напечатать перед вершиной по формуле: (количество потомков вершины - 1) * (количество знакомест, занимаемых одной вершиной дерева + временная переменная 1 + временная переменная 2) / 2 + временная переменная 1.

9.6.2.1.4. Вычисляем количество пробелов, которые нужно напечатать после вершины по формуле: количество пробелов, которые нужно напечатать перед вершиной - временная переменная 1 + временная переменная 2.

9.6.3. Вывод вычисленного количества пробелов перед вершиной.

9.6.4. Задаем форматную строку для вывода одного значения вершины дерева. Форматная строка имеет вид «%x<x>d», где x<x> - одно- или двухразрядное десятичное целое число знакомест, занимаемых одним значением вершины дерева.

9.6.5. Вывод значений вершины (если значения вершины нулевые выводятся символ '*') и вычисленного количества пробелов после вершины.

Реализация методов класса **Node**, вложенного в шаблонный класс **BSTree**:

10. Конструктор с параметром `Node(const Type &)`. Инициализирует поле данных вершины значением параметра и значениями `NULL` указатели на потомков.

11. Получение данных в текущей вершине дерева `Type GetData()`. Возвращает значение данных вершины.

12. Запись данных в текущую вершину дерева `void SetData(const Type &)`. Присваивает полю данных вершины значение параметра.

13. Получение указателя на вершину левого поддерева текущей вершины дерева `Node * GetLeft()`. Возвращает адрес вершины левого потомка.

14. Запись указателя на вершину левого поддерева в текущую вершину дерева `void SetLeft(Node *)`. Присваивает полю указателя на левого потомка вершины значение параметра.

15. Получение указателя на вершину правого поддерева текущей вершины дерева `Node * GetRight()`. Возвращает адрес вершины правого потомка.

16. Запись указателя на вершину правого поддерева в текущую вершину дерева `void SetRight(Node *)`. Присваивает полю указателя на правого потомка вершины значение параметра.

4. Описание работы программы

После запуска программы на экране появляется меню, показанное на Рис. 1.

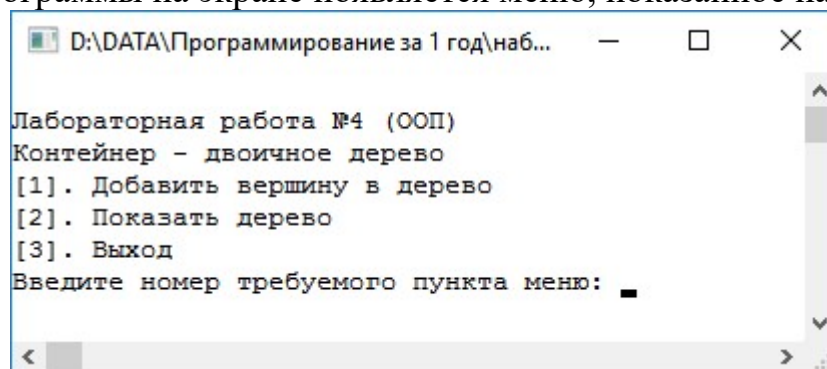


Рис. 1 Главное меню программы.

При выборе пункта 1 будет предложено ввести целочисленное значение данных добавляемой вершины дерева. Рис. 2.

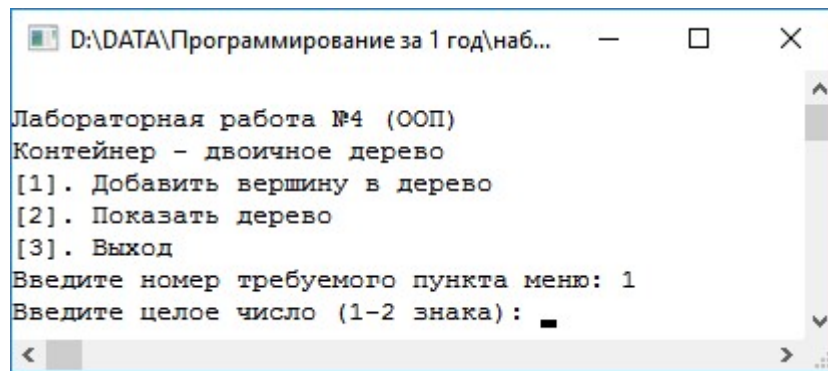


Рис. 2 Результат выбора пункта 2 главного меню программы.

После ввода значения данных вершины будет выведено сообщение «Добавлено» и главное меню программы. Рис. 3.

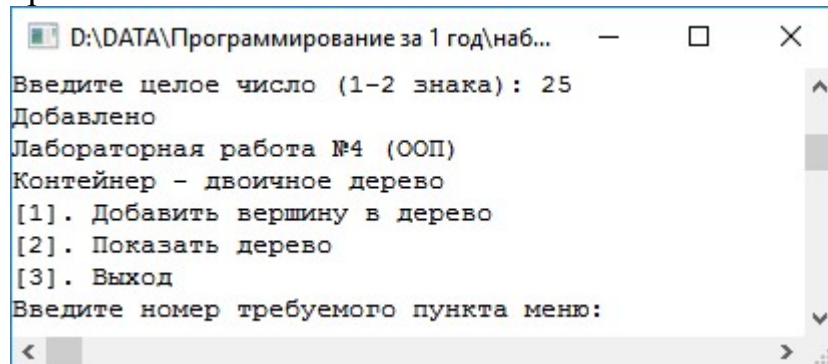


Рис. 3 Результат ввода данных вершины дерева.

При выборе пункта 2 главного меню программы на экран будет выведено количество вершин, дерево, элементы которого хранятся в созданном объекте, и главное меню программы. Рис. 4.

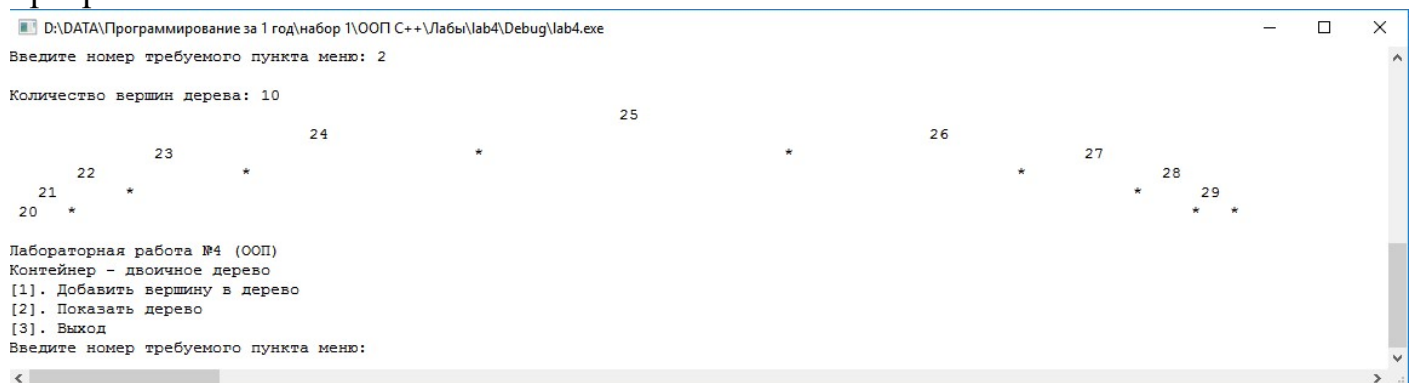


Рис. 4 Результат выбора пункта 2 главного меню программы.

При выборе пункта 3 главного меню выводится сообщение о количестве неудаленных вершин дерева Рис. 5.

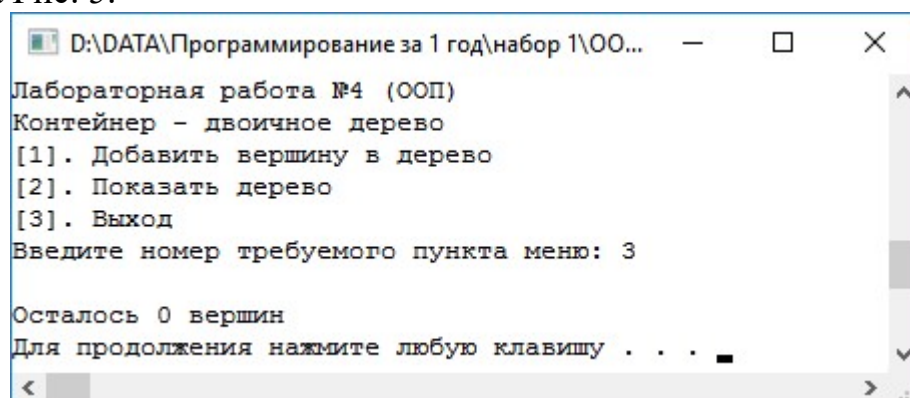


Рис. 5 Результат выбора пункта 3 главного меню программы.

После нажатия клавиши происходит выход из программы.

5. Выводы

В результате выполнения данной лабораторной работы, получен практический опыт работы с шаблонами классов и методов. Целью данной работы является знакомство с шаблонными классами и методами.

В данной лабораторной работе создан шаблонный класс дерева с вложенным классом вершины. В шаблонном классе определены данные и методы:

- Вложенный класс вершины дерева, в котором определены:
 - Поле данных вершины дерева.
 - Указатель на вершину левого поддерева.
 - Указатель на вершину правого поддерева.
 - Конструктор с параметром.
 - Конструктор копирования.
 - Метод получения данных в текущей вершине дерева.
 - Метод записи данных в текущую вершину дерева.
 - Метод получения указателя на вершину левого поддерева текущей вершины дерева.
 - Метод записи указателя на вершину левого поддерева в текущую вершину дерева.
 - Метод получения указателя на вершину правого поддерева текущей вершины дерева.
 - Метод записи указателя на вершину правого поддерева в текущую вершину дерева.
- Указатель на корень дерева.
- Количество вершин дерева.
- Конструктор по умолчанию.
- Деструктор.
- Метод освобождения памяти из-под вершин дерева.
- Метод вставки вершины дерева.
- Метод получения количества вершин дерева.
- Метод получения указателя на корень дерева.
- Метод вычисления высоты дерева с учетом веток к нулевым вершинам.
- Метод вывода дерева на экран.
- Метод вывода одного уровня дерева на экран.

Файлы программы находятся в репозитории по адресу: <https://github.com/broitman-eugeny/lab4>.

6. Код программы с комментариями

“lab4.h”

```
#pragma once
#include <iostream>
#include <Windows.h>
const int ValChPlacesAmount = 2; //Количество знакомест для отображения одного значения данных на
экране
const int OutSpaceChPlacesAmount = 2; //Количество знакомест для разделителя между вершинами нижнего
уровня
const int InSpaceChPlacesAmount = 1; //Количество знакомест для разделителя между значениями данных
одной вершины
//Шаблон класса дерева двоичного поиска
template <class Type>
class BSTree
{
    //Вложенный класс вершины дерева
    class Node
    {
        //Данные вершины
        Type Data;
        //Указатель на вершину левого поддерева
        Node *Left;
        //Указатель на вершину правого поддерева
        Node *Right;
    public:
        Node(const Type &);
        Node(const Node &);
        Type GetData(); //Получение данных в текущей вершине дерева
        void SetData(const Type &); //Запись данных в текущую вершину дерева
        Node * GetLeft(); //Получение указателя на вершину левого поддерева текущей вершины де-
рева
        void SetLeft(Node *); //Запись указателя на вершину левого поддерева в текущую вершину
дерева
        Node * GetRight(); //Получение указателя на вершину правого поддерева текущей вершины
дерева
        void SetRight(Node *); //Запись указателя на вершину правого поддерева в текущую верши-
ну дерева
    };
    //Указатель на корень дерева
    Node *Root;
    //Количество вершин дерева
    int Count;
    public:
        //Конструктор по умолчанию
        BSTree();
        ~BSTree(); //Деструктор
        Node* ClearAll(Node*); //Удаление дерева
        Node * Paste(Node *, const Type &); //Вставка вершины дерева
        int GetCount(); //Получение количества вершин дерева
        Node *GetRoot(); //Получение указателя на корень дерева
        int TreeHeight(Node * LocalRoot); //Вычисляет высоту дерева с учетом веток к нулевым вершинам
        void Show(Node * LocalRoot); //Вывод дерева на экран
        void ShowLevel(Node * LocalRoot, int ReqLevel, int Height, int TempLevel); //Вывод одного
уровня дерева на экран
    };
    //Конструктор с параметром
    template<class Type>
    BSTree<Type>::Node::Node(const Type &T) : Data(T), Left(NULL), Right(NULL)
    {
    }
    //Конструктор копирования. typename необходимо (см.
https://msdn.microsoft.com/query/dev14.query?appId=Dev14IDEF1&l=RU-RU&k=k\(C4346\)&rd=true)
    template<class Type>
    BSTree<Type>::Node::Node(const typename BSTree<Type>::Node &T) : Data(T.Data), Left(T.Left),
    Right(T.Right)
    {
    }
    template<class Type>
    Type BSTree<Type>::Node::GetData()
```

```

{
    return Data;
}
template<class Type>
void BSTree<Type>::Node::SetData(const Type &T)
{
    Data = T;
}
template<class Type>
typename BSTree<Type>::Node * BSTree<Type>::Node::GetLeft()//typename необходимо (см.
https://msdn.microsoft.com/query/dev14.query?appId=Dev14IDEF1&l=RU-RU&k=k\(C4346\)&rd=true)
{
    return Left;
}
template<class Type>
void BSTree<Type>::Node::SetLeft(typename BSTree<Type>::Node *T)
{
    Left = T;
}
template<class Type>
typename BSTree<Type>::Node * BSTree<Type>::Node::GetRight()//typename необходимо (см.
https://msdn.microsoft.com/query/dev14.query?appId=Dev14IDEF1&l=RU-RU&k=k\(C4346\)&rd=true)
{
    return Right;
}
template<class Type>
void BSTree<Type>::Node::SetRight(typename BSTree<Type>::Node *T)
{
    Right = T;
}
//Конструктор по умолчанию
template<class Type>
BSTree<Type>::BSTree() : Root(NULL), Count(0)
{
}
//Деструктор
template<class Type>
BSTree<Type>::~BSTree()
{
    ClearAll(Root);//Удалить все вершины дерева
    std::cout << std::endl << "Осталось " << GetCount() << " вершин" << std::endl;
}
//Удаление всех вершин дерева
//LocalRoot - указатель на текущую вершину дерева, а для первого вызова - указатель на корень дерева
template<class Type>
typename BSTree<Type>::Node * BSTree<Type>::ClearAll(typename BSTree<Type>::Node *LocalRoot)
{
    if (LocalRoot == NULL)//Дошли до конца ветки
    {
        return NULL;
    }
    BSTree<Type>::Node *LocalLeft, *LocalRight;
    LocalLeft = ClearAll(LocalRoot->GetLeft());//Идем в конец левой ветки
    LocalRight= ClearAll(LocalRoot->GetRight());//Идем в конец правой ветки
    //Дошли до конца обеих веток
    delete LocalRoot;
    Count--;
    return NULL;
}
//Вставка вершины дерева
//Ищет место вставки вершины по критерию упорядоченности (левое поддереву должно иметь значения дан-
ных
//строго меньше значения данных родителя, а правое поддереву должно иметь значения данных больше,
//либо равные значению данных родителя), устанавливает ссылки на соответствующие вершины.
//LocalRoot - указатель на текущую вершину дерева, а для первого вызова - указатель на корень дерева
//T - значение данных вершины
//Возвращает указатель на корень дерева
template<class Type>

```

```

typename BSTree<Type>::Node * BSTree<Type>::Paste(typename BSTree<Type>::Node *LocalRoot, const Type
&T)//typename необходимо (см. https://msdn.microsoft.com/query/dev14.query?appId=Dev14IDEF1&l=RU-
RU&k=k(C4346)&rd=true)
{
    if (Count == 0)//Дерево пустое
    {
        Root = new Node(T);//Создание вершины со значением данных T и указателями на потомков
        Count = 1;
        return Root;
    }
    else//Дерево не пустое
    {
        if (LocalRoot == NULL)//Текущая вершина пустая
        {
            Count++;
            return new Node(T);//Создание вершины со значением данных T и указателями на
            потомков NULL
        }
        else//Текущая вершина занята. Рекурсивно ищем подходящую свободную вершину
        {
            if (LocalRoot->GetData() > T)//Если данные в текущей вершине дерева строго боль-
            ше вставляемого значения
            {
                //Ищем в левом поддереве подходящую пустую вершину
                LocalRoot->SetLeft(Paste(LocalRoot->GetLeft(), T));
            }
            else//Данные в текущей вершине дерева не больше вставляемого значения
            {
                //Ищем в правом поддереве подходящую пустую вершину
                LocalRoot->SetRight(Paste(LocalRoot->GetRight(), T));
            }
            return LocalRoot;
        }
    }
}

template<class Type>
int BSTree<Type>::GetCount()
{
    return Count;
}

template<class Type>
typename BSTree<Type>::Node * BSTree<Type>::GetRoot()
{
    return Root;
}

//Функция вычисляет высоту дерева с учетом веток к нулевым вершинам
//LocalRoot - указатель на текущую вершину дерева, а для первого вызова - указатель на корень дерева
template<class Type>
int BSTree<Type>::TreeHeight(typename BSTree<Type>::Node * LocalRoot)
{
    if (LocalRoot == NULL)//Дошли до конца ветви
    {
        return 0;
    }
    int LeftHeight, RightHeight;//Высоты поддеревьев
    LeftHeight = TreeHeight(LocalRoot->GetLeft());//Высота левого поддерева
    RightHeight = TreeHeight(LocalRoot->GetRight());//Высота правого поддерева
    return ((LeftHeight>RightHeight) ? LeftHeight : RightHeight) + 1;//Максимальная+1(текущая)
}

//Функция отображает дерево
//LocalRoot - указатель на корень дерева
template<class Type>
void BSTree<Type>::Show(typename BSTree<Type>::Node * LocalRoot)
{
    int Height = TreeHeight(LocalRoot);//Высота дерева с учетом нулевых веток
    printf("\n");
    for (int i = 0; i < Height; i++)
    {

```

```

        ShowLevel(LocalRoot, i, Height, 0); //Отображает один уровень дерева
        printf("\n");
    }
}
//Функция отображает один уровень дерева
//LocalRoot - указатель на текущую вершину дерева, а для первого вызова - указатель на корень дерева
//ReqLevel - номер печатаемого уровня
//Height - высота дерева, включающая ветки к нулевым вершинам
//TempLevel - текущий уровень (для поиска вершин печатаемого уровня)
template<class Type>
void BSTree<Type>::ShowLevel(typename BSTree<Type>::Node * LocalRoot, int ReqLevel, int Height, int
TempLevel)
{
    char FormatS[5];
    FormatS[0] = '%';
    FormatS[1] = (ValChPlacesAmount > 9) ? '0' + ValChPlacesAmount / 10 : '0' + ValChPlacesAmount;
    FormatS[2] = (ValChPlacesAmount > 9) ? '0' + ValChPlacesAmount % 10 : 's';
    FormatS[3] = (ValChPlacesAmount > 9) ? 's' : '\0';
    FormatS[4] = '\0';
    int Temp1 = OutSpaceChPlacesAmount / 2; //Временная переменная 1
    Temp2 = OutSpaceChPlacesAmount - Temp1; //Временная переменная 2
    SpacesBeforeNode = Temp1; //Количество пробелов, которые нужно напечатать перед верши-
ной текущего уровня
    SpacesAfterNode = Temp2; //Количество пробелов, которые нужно напечатать после вершины
текущего уровня
    W = ValChPlacesAmount; //Количество знакомест, занимаемых одной вершиной дерева
    if (TempLevel < ReqLevel) //Пока не достигли требуемого уровня
    {
        if (LocalRoot != NULL) //Не достигли конца ветви
        {
            ShowLevel(LocalRoot->GetLeft(), ReqLevel, Height, TempLevel + 1); //Переход к
следующему уровню в левом поддереве
            ShowLevel(LocalRoot->GetRight(), ReqLevel, Height, TempLevel + 1); //Переход к
следующему уровню в правом поддереве
        }
        else //достигли конца ветви. Необходимо напечатать пробелы в запрашиваемом уровне (тек-
стовый курсор находится в запрашиваемом уровне), чтобы восполнить отсутствующие элементы из-за нуле-
вых вершин в текущем уровне и сохранить симметрию дерева
        {
            for (int i = Height - 2; i >= TempLevel; i--)
            {
                Temp1 = SpacesBeforeNode; //Значение со следующего уровня
                Temp2 = SpacesAfterNode; //Значение со следующего уровня
                //Количество пробелов, которые
нужно напечатать перед вершиной
                SpacesBeforeNode = 2 * (W + Temp1 + Temp2) / 2 - W / 2;
                //Количество пробелов, которые нужно напечатать после вершины
                SpacesAfterNode = SpacesBeforeNode + W / 2 - (W - W / 2); //Т.к. W м.б.
нечетным и W/2 < половины W
            }
            //Печать пробелов перед вершиной
            for (int i = 0; i < SpacesBeforeNode; i++)
            {
                printf(" ");
            }
            printf(FormatS, " "); //Печать пробелов вместо отсутствующих вершин, чтобы сохра-
нить симметрию

            //Печать пробелов после вершины
            for (int i = 0; i < SpacesAfterNode; i++)
            {
                printf(" ");
            }
        }
    }
    else //Достигли требуемого уровня
    {
        for (int i = Height - 2; i >= TempLevel; i--)
        {
            Temp1 = SpacesBeforeNode;

```

```

        Temp2 = SpacesAfterNode;
        //Количество пробелов, которые нужно напечатать перед вершиной
        SpacesBeforeNode = 2 * (W + Temp1 + Temp2) / 2 - W / 2;
        //Количество пробелов, которые нужно напечатать после вершины
        SpacesAfterNode = SpacesBeforeNode + W / 2 - (W - W / 2); //Т.к. W м.б. нечетным
и W/2 < половины W
    }
    //Печать пробелов перед вершиной
    for (int i = 0; i < SpacesBeforeNode; i++)
    {
        printf(" ");
    }
    char Format[5];
    Format[0] = '%';
    Format[1] = (ValChPlacesAmount > 9) ? '0' + ValChPlacesAmount / 10 : '0' +
ValChPlacesAmount;
    Format[2] = (ValChPlacesAmount > 9) ? '0' + ValChPlacesAmount % 10 : 'd';
    Format[3] = (ValChPlacesAmount > 9) ? 'd' : '\0';
    Format[4] = '\0';
    //Печать значений вершины
    if (LocalRoot != NULL) //Вершина имеется (не конец ветви)
    {
        printf(Format, LocalRoot->GetData());
    }
    else //Нулевая вершина (конец ветви)
    {
        printf(FormatS, "*");
    }
    //Печать пробелов после вершины
    for (int i = 0; i < SpacesAfterNode; i++)
    {
        printf(" ");
    }
}
}
//Функция меню
void Menu();
“main.cpp”
//Лабораторная работа №4. Разработка шаблона
//Вариант №5.
//Необходимо разработать шаблон класса, реализующий структуру данных (контейнер).
//Параметры шаблона передаются через угловые скобки, и позволяют строить контейнер на статическом массиве.
//Шаблон, помимо структуры хранения, должен включать в себя:
//1) Функции добавления значения (оператор индексирования для массивов, Push для стека, Into, Add и т.п.).
//2) Функции получения значения.
//3) Дополнительные функции (проверка количества элементов, переполнения и т.п.).
//Все функции должны проверять корректность входных значений, проверять переполнение
//(в общем, обеспечивать полностью корректную работу контейнера).
//В рамках данной работы, рекомендуется хранить непосредственно объекты (а не указатели на них).
//Пользовательский интерфейс должен продемонстрировать работу шаблона.
//В простейшем случае, это занесение ряда значений, а потом их получение с выводом на экран.
//Демонстрацию работы шаблона можно провести на любом простом или сложном типе данных.
//ОБЯЗАТЕЛЬНО продемонстрировать работу механизма обработки переполнений.
//Тип контейнера - двоичное дерево.
#include "lab4.h"
void main()
{
    SetConsoleCP(1251); //Ввод русских букв
    SetConsoleOutputCP(1251); //Вывод русских букв
    Menu();
    system("pause");
}
“Menu.cpp”
#include "lab4.h"
//Функция отображения меню
void ShowMenu()
{

```

```

        std::cout << std::endl << "Лабораторная работа №4 (ООП)" << std::endl << "Контейнер - двоич-
ное дерево";
        std::cout << std::endl << "[1]. Добавить вершину в дерево";
        std::cout << std::endl << "[2]. Показать дерево";
        std::cout << std::endl << "[3]. Выход";
        std::cout << std::endl << "Введите номер требуемого пункта меню: ";
    }
    //Функция проверки ввода из потока std::cin целочисленного значения
    //Value - ссылка на переменную в которую осуществляется ввод
    //LeftBorder - левая граница области определения вводимой переменной
    //LeftIncluded - признак включения в область определения переменной левой границы
    //RightBorder - правая граница области определения вводимой переменной
    //RightIncluded - признак включения в область определения переменной правой границы
    void CheckCinInt(int &Value, const int LeftBorder, const bool LeftIncluded, const int RightBorder,
const bool RightIncluded)
    {
        while ((LeftIncluded == true) ? (Value<LeftBorder) : (Value <= LeftBorder) || (RightIncluded
== true) ? (Value>RightBorder) : (Value >= RightBorder) || std::cin.fail())
        {
            std::cin.clear();
            std::cin.ignore(std::cin.rdbuf()->in_avail()); //Очистка буфера
            std::cout << std::endl << "Значение введено неверно, введите еще раз: ";
            std::cin >> Value;
        }
        std::cin.clear();
        std::cin.ignore(std::cin.rdbuf()->in_avail()); //Очистка буфера если введено вещественное чис-
ло попадающее в заданный диапазон
    }
    //Функция меню
    void Menu()
    {
        BSTree<int> TreeInt10; //Контейнер-дерево для целых чисел на 10 вершин
        int T;
        int C = -1;
        while (C != 3) //3 - выход
        {
            ShowMenu();
            std::cin >> C;
            CheckCinInt(C, 1, true, 3, true); //проверка корректности ввода
            switch (C)
            {
                case 1: //Добавить вершину в дерево
                    std::cout << "Введите целое число (1-2 знака): ";
                    std::cin >> T;
                    TreeInt10.Paste(TreeInt10.GetRoot(), T);
                    std::cout << "Добавлено";
                    break;
                case 2: //Показать дерево
                    std::cout << std::endl << "Количество вершин дерева: " << TreeInt10.GetCount();
                    TreeInt10.Show(TreeInt10.GetRoot()); //Вывод дерева на экран
                    break;
                case 3: //Выход
                    break;
            } //switch (C)
        } //while (C != 3) //3 - выход
    }
}

```