|  |
| --- |
| #include<iostream> |
|  | использование пространства имен std; |
|  |  |
|  | #определить вкладку "\t" |
|  |  |
|  | дерево классов |
|  | { |
|  | элемент класса |
|  | { |
|  | int Data; |
|  | Элемент \*pLeft, \*pRight; |
|  | Публично: |
|  | Element(int Data, Element\* pLeft = nullptr, Element\* pRight = nullptr) :Data(данные), pLeft(плефт), pRight(прайхт) |
|  | { |
|  | #ifdef DEBUG |
|  | cout < < "EConstructor:\t" << this <; |
|  | #endif // DEBUG |
|  | } |
|  | ~Элемент() |
|  | { |
|  | #ifdef DEBUG |
|  | cout << "EDestructor:\t" << this <; |
|  | #endif // DEBUG |
|  |  |
|  | } |
|  | bool is\_leaf() |
|  | { |
|  | возврат pLeft == pRight; |
|  | } |
|  | дерево классов друзей; |
|  | }\*Корень; |
|  | Публично: |
|  | Элемент\* getRoot() |
|  | { |
|  | верните это->>Root; |
|  | } |
|  | Дерево() :корень(nullptr) |
|  | { |
|  | #ifdef DEBUG |
|  | cout < < "TConstructor:\t" << this <; |
|  | #endif // DEBUG |
|  | } |
|  | Tree(const Tree& other) |
|  | { |
|  | copy(other.Root); |
|  | cout << "CopyConstructor:" << this << endl; |
|  | } |
|  | Tree(const initializer\_list<int>& il) :Tree() |
|  | { |
|  | for (int const\* it = il.begin(); it != il.end(); it++) |
|  | insert(\*it, this->Root); |
|  | } |
|  | ~Tree() |
|  | { |
|  | clear(this->Root); |
|  | #ifdef DEBUG |
|  | cout << "TDestructor:\t" << this << endl; |
|  | #endif // DEBUG |
|  | } |
|  | void insert(int Data) |
|  | { |
|  | insert(Data, this->Root); |
|  | } |
|  | void erase(int Data) |
|  | { |
|  | erase(Data, this->Root); |
|  | } |
|  | int minValue() |
|  | { |
|  | if (this->Root) |
|  | return minValue(this->Root); |
|  | else |
|  | return 0; |
|  | } |
|  | int maxValue() |
|  | { |
|  | return this->Root ? maxValue(this->Root) : 0; |
|  | } |
|  | int size() |
|  | { |
|  | return this->Root ? count(this->Root) : 0; |
|  | } |
|  | int sum() |
|  | { |
|  | return sum(this->Root); |
|  | } |
|  | double avg() |
|  | { |
|  | return (double)sum(this->Root) / count(this->Root); |
|  | } |
|  | void clear() |
|  | { |
|  | clear(this->Root); |
|  | this->Root = nullptr; |
|  | } |
|  | void print() |
|  | { |
|  | print(this->Root); |
|  | cout << endl; |
|  | } |
|  | private: |
|  | void insert(int Data, Element\* Root)//Root - корневой элемент поддерева (ветки) |
|  | { |
|  | if (this->Root == nullptr)this->Root = new Element(Data); |
|  | if (Root == nullptr)return; //Если зашли в какую-либо ветку, а она оказалась пустой - сразу же выходим из нее. |
|  | if (Data < Root->Data) |
|  | { |
|  | if (Root->pLeft == nullptr) |
|  | Root->pLeft = new Element(Data); |
|  | else |
|  | insert(Data, Root->pLeft); |
|  | } |
|  | else //if (Data > Root->Data) |
|  | { |
|  | if (Root->pRight)insert(Data, Root->pRight); |
|  | else Root->pRight = new Element(Data); |
|  | } |
|  | } |
|  | void erase(int Data, Element\*& Root) |
|  | { |
|  | if (Root == nullptr)return; |
|  | erase(Data, Root->pLeft); |
|  | erase(Data, Root->pRight); |
|  | if (Data == Root->Data) |
|  | { |
|  | if (Root->is\_leaf()) **ЛИСТ** |
|  | { |
|  | delete Root; |
|  | Root = nullptr; |
|  | } |
|  | else |
|  | { |
|  | if (Root->pLeft) **ЕСТЬ ЛЕВАЯ ВЕТВЬ** |
|  | { |
|  | Root->Data = maxValue(Root->pLeft); |
|  | erase(maxValue(Root->pLeft), Root->pLeft); |
|  | } |
|  | Else **ЕСТЬ ПРАВАЯ ВЕТВЬ** |
|  | { |
|  | Root->Data = minValue(Root->pRight); |
|  | erase(minValue(Root->pRight), Root->pRight); |
|  | } |
|  | } |
|  | //return; |
|  | } |
|  |  |
|  | } |
|  | int minValue(Element\* Root) |
|  | { |
|  | if (Root->pLeft == nullptr)return Root->Data; |
|  | else return minValue(Root->pLeft); |
|  | } |
|  | int maxValue(Element\* Root) |
|  | { |
|  | /\*if (Root->pRight)return maxValue(Root->pRight); |
|  | else return Root->Data;\*/ |
|  | return Root->pRight ? maxValue(Root->pRight) : Root->Data; |
|  | } |
|  | int count(Element\* Root) |
|  | { |
|  | if (Root == nullptr)return 0; |
|  | /\*if (Root->is\_leaf())return 1; |
|  | else return count(Root->pLeft) + count(Root->pRight) + 1;\*/ |
|  | return Root->is\_leaf() ? 1 : count(Root->pLeft) + count(Root->pRight) + 1; |
|  | } |
|  | int sum(Element\* Root) |
|  | { |
|  | //если (Root == nullptr)возвращает 0; |
|  | вернуть корень ? sum(Root->pLeft) + sum(Root->pRight) + Root->Data : 0; |
|  | } |
|  | void copy(элемент\* корень) |
|  | { |
|  | if (Root == nullptr)return; |
|  | вставить(корень->>данные, это->>корень); |
|  | copy(Root->pLeft); |
|  | копировать(Root->>pRight); |
|  | } |
|  | void clear(элемент\* корень) |
|  | { |
|  | if (Root == nullptr)return; |
|  | очистить(корень->>складка); |
|  | clear(Root->pRight); |
|  | удалить корень; |
|  | } |
|  | void print(элемент\* корень) |
|  | { |
|  | if (Root == nullptr)return; |
|  | печать(корень->>складка); |
|  | cout << Root->Data << tab; |
|  | print(Root->pRight); |
|  | } |
|  | }; |
|  |  |
|  | //#define ERASE\_CHECK |
|  |  |
|  | пустота главная() |
|  | { |
|  | setlocale(LC\_ALL, "русский"); |
|  | int n; //Размер дерева |
|  | Дерево Т = { 50, 25, 16, 32, 8, 85, 64, 62, 80, 91, 98 }; |
|  | /\*cout << "Введите количество элементов: "; cin >> n; |
|  | for (int i = 0; t.size() < n, i) |
|  | { |
|  | т. вставить(rand() % 100); |
|  | }\*/ |
|  | t.печать(); |
|  | cout << "Минимальное значение в дереве: " << t.minValue() << endl; |
|  | cout << "Минимальное значение в дереве: " << t.maxValue() << endl; |
|  | cout << "Количетво элементов дерева: " << t.size() << endl; |
|  | cout << "Сумма элементов дерева: " << t.sum() << endl; |
|  | cout << "Среднее арифметическое элементов дерева: " << t.avg() << endl; |
|  | int value; cout << "Введите удаляемое значение: "; cin >> value; |
|  | t.erase(значение); |
|  | t.печать(); |
|  | cout << "Минимальное значение в дереве: " << t.minValue() << endl; |
|  | cout << "Минимальное значение в дереве: " << t.maxValue() << endl; |
|  | cout << "Количетво элементов дерева: " << t.size() << endl; |
|  | cout << "Сумма элементов дерева: " << t.sum() << endl; |
|  | cout << "Среднее арифметическое элементов дерева: " << t.avg() << endl; |
|  |  |
|  | #ifdef ERASE\_CHECK |
|  | Tree t2 = { 50, 25, 16, 32, 8, 85, 64, 62, 80, 91, 98 }; |
|  | t2.print(); |
|  | int value; cout << "Введите удаляемое значение: "; cin >> value; |
|  | t2.erase(value); |
|  | t2.print(); |
|  | #endif // ERASE\_CHECK |
|  |  |
|  | } |