**Лабораторная работа № 12. Бинарные кучи**

|  |  |
| --- | --- |
| 1. Пункты 1, 2, 3 содержат программный код проекта, в котором представлена бинарная куча, реализованная в виде *массива*.  В правой части данного пункта записан *заголовочный файл* **Heap.h**.  Написать комментарии к программному коду. | #pragma once  // Объявление структуры AAA с целочисленным полем x и методом print()  struct AAA  {  int x; // Целочисленное поле для хранения данных  void print(); // Метод для вывода значения x на экран  };  // Пространство имен для работы с кучей  namespace heap  {  // Перечисление для сравнения элементов (LESS - меньше, EQUAL - равно, GREAT - больше)  enum CMP  {  LESS = -1,  EQUAL = 0,  GREAT = 1  };  // Структура, представляющая кучу (пирамиду)  struct Heap  {  int size; // Текущий размер кучи  int maxSize; // Максимальный размер кучи  void\*\* storage; // Массив для хранения элементов (указателей на void)  CMP(\*compare)(void\*, void\*); // Указатель на функцию сравнения элементов  // Конструктор кучи  Heap(int maxsize, CMP(\*f)(void\*, void\*))  {  size = 0;  storage = new void\* [maxSize = maxsize]; // Выделение памяти под массив  compare = f; // Инициализация функции сравнения  };  // Методы кучи:  int left(int ix); // Получить индекс левого потомка  int right(int ix); // Получить индекс правого потомка  int parent(int ix); // Получить индекс родителя  // Проверка, заполнена ли куча  bool isFull() const { return (size >= maxSize); };  // Проверка, пуста ли куча  bool isEmpty() const { return (size <= 0); };  // Сравнение элементов через функцию сравнения  bool isLess(void\* x1, void\* x2) const { return compare(x1, x2) == LESS; };  bool isGreat(void\* x1, void\* x2) const { return compare(x1, x2) == GREAT; };  bool isEqual(void\* x1, void\* x2) const { return compare(x1, x2) == EQUAL; };  void swap(int i, int j); // Обмен элементов местами  void heapify(int ix); // Восстановление свойств кучи  void insert(void\* x); // Вставка нового элемента  void\* extractMax(); // Извлечение максимального элемента  void scan(int i) const; // Вывод кучи на экран  };  // Функция создания кучи  Heap create(int maxsize, CMP(\*f)(void\*, void\*));  };  #include "Heap.h"  #include <iostream>  #include <iomanip>  // Реализация метода print() для структуры AAA  void AAA::print()  {  std::cout << x; // Просто выводим значение x  }  // Реализация методов кучи  namespace heap  {  // Создание новой кучи  Heap create(int maxsize, CMP(\*f)(void\*, void\*))  {  return \*(new Heap(maxsize, f)); // Создаем и возвращаем новую кучу  }  // Получение индекса левого потомка  int Heap::left(int ix)  {  return (2 \* ix + 1 >= size) ? -1 : (2 \* ix + 1); // -1 если потомка нет  }  // Получение индекса правого потомка  int Heap::right(int ix)  {  return (2 \* ix + 2 >= size) ? -1 : (2 \* ix + 2); // -1 если потомка нет  }  // Получение индекса родителя  int Heap::parent(int ix)  {  return (ix + 1) / 2 - 1; // Формула для расчета индекса родителя  }  // Обмен элементов местами  void Heap::swap(int i, int j)  {  void\* buf = storage[i]; // Временное хранение элемента i  storage[i] = storage[j]; // Перемещение элемента j на место i  storage[j] = buf; // Перемещение элемента i на место j  }  // Восстановление свойств кучи (просеивание вниз)  void Heap::heapify(int ix)  {  int l = left(ix), r = right(ix), irl = ix; // Индексы левого, правого потомков и наибольшего элемента  if (l > 0) // Если левый потомок существует  {  if (isGreat(storage[l], storage[ix])) irl = l; // Если левый больше текущего  if (r > 0 && isGreat(storage[r], storage[irl])) irl = r; // Если правый больше наибольшего  if (irl != ix) // Если наибольший не текущий  {  swap(ix, irl); // Меняем местами  heapify(irl); // Рекурсивно вызываем для нового положения  }  }  }  // Вставка нового элемента  void Heap::insert(void\* x)  {  int i;  if (!isFull()) // Если куча не заполнена  {  storage[i = ++size - 1] = x; // Добавляем элемент в конец  // Просеивание вверх  while (i > 0 && isLess(storage[parent(i)], storage[i]))  {  swap(parent(i), i); // Меняем с родителем  i = parent(i); // Переходим к родителю  }  }  }  // Извлечение максимального элемента  void\* Heap::extractMax()  {  void\* rc = nullptr;  if (!isEmpty()) // Если куча не пуста  {  rc = storage[0]; // Максимальный элемент - корень  storage[0] = storage[size - 1]; // Перемещаем последний элемент в корень  size--; // Уменьшаем размер  heapify(0); // Восстанавливаем свойства кучи  }  return rc; // Возвращаем максимальный элемент  }  // Вывод кучи на экран в виде дерева  void Heap::scan(int i) const  {  int probel = 20; // Начальное количество пробелов для форматирования  std::cout << '\n';  if (size == 0) // Если куча пуста  std::cout << "Куча пуста";  // Вывод элементов с форматированием  for (int u = 0, y = 0; u < size; u++)  {  std::cout << std::setw(probel + 10) << std::setfill(' ');  ((AAA\*)storage[u])->print(); // Вывод элемента  if (u == y) // Переход на новую строку для нового уровня дерева  {  std::cout << '\n';  if (y == 0)  y = 2;  else  y += y \* 2;  }  probel /= 2; // Уменьшение отступа для следующих уровней  }  std::cout << '\n';  }  }  #include "Heap.h"  #include <iostream>  using namespace std;  // Функция сравнения для структуры AAA  heap::CMP cmpAAA(void\* a1, void\* a2)  {  #define A1 ((AAA\*)a1) // Приведение типа для удобства  #define A2 ((AAA\*)a2)  heap::CMP rc = heap::EQUAL; // По умолчанию элементы равны  if (A1->x > A2->x)  rc = heap::GREAT; // Первый элемент больше  else if (A2->x > A1->x)  rc = heap::LESS; // Первый элемент меньше  return rc;  #undef A2  #undef A1  }  // Главная функция с меню для тестирования кучи  int main()  {  setlocale(LC\_ALL, "rus"); // Установка русской локали  int k, choice;  heap::Heap h1 = heap::create(30, cmpAAA); // Создание кучи  // Бесконечный цикл меню  for (;;)  {  cout << "1 - вывод кучи на экран" << endl;  cout << "2 - добавить элемент" << endl;  cout << "3 - удалить максимальный элемент" << endl;  cout << "0 - выход" << endl;  cout << "сделайте выбор" << endl;  cin >> choice;  switch (choice)  {  case 0: exit(0); // Выход из программы  case 1: h1.scan(0); // Вывод кучи  break;  case 2:  {  AAA\* a = new AAA; // Создание нового элемента  cout << "введите ключ" << endl;  cin >> k;  a->x = k; // Установка значения  h1.insert(a); // Вставка в кучу  }  break;  case 3: h1.extractMax(); // Удаление максимального элемента  break;  default: cout << endl << "Введена неверная команда!" << endl;  }  }  return 0;  } |
| 5.  В проект добавить следующие функции: удаление минимального **extractMin**; удаление i-ого элемента **extractI**; объединение **unionHeap** двух куч в одну. | #include "Heap.h"  #include <iostream>  #include <iomanip>  void AAA::print()  {  std::cout << x;  }  namespace heap  {  Heap create(int maxsize, CMP(\*f)(void\*, void\*))  {  return Heap(maxsize, f);  }  int Heap::left(int ix)  {  int idx = 2 \* ix + 1;  return (idx < size) ? idx : -1;  }  int Heap::right(int ix)  {  int idx = 2 \* ix + 2;  return (idx < size) ? idx : -1;  }  int Heap::parent(int ix)  {  return (ix > 0) ? (ix - 1) / 2 : -1;  }  void Heap::swap(int i, int j)  {  void\* buf = storage[i];  storage[i] = storage[j];  storage[j] = buf;  }  void Heap::heapify(int ix)  {  int largest = ix;  int l = left(ix), r = right(ix);  if (l != -1 && isGreat(storage[l], storage[largest]))  largest = l;  if (r != -1 && isGreat(storage[r], storage[largest]))  largest = r;  if (largest != ix)  {  swap(ix, largest);  heapify(largest);  }  }  void Heap::insert(void\* x)  {  if (isFull())  return;  int i = size++;  storage[i] = x;  int p = parent(i);  while (p != -1 && isLess(storage[p], storage[i]))  {  swap(p, i);  i = p;  p = parent(i);  }  }  void\* Heap::extractMax()  {  if (isEmpty())  return nullptr;  void\* rc = storage[0];  storage[0] = storage[--size];  heapify(0);  return rc;  }  void Heap::scan(int i) const //Вывод значений элементов на экран  {  int probel = 20;  std::cout << '\n';  if (size == 0)  std::cout << "Куча пуста";  for (int u = 0, y = 0; u < size; u++)  {  std::cout << std::setw(probel + 10) << std::setfill(' ');  ((AAA\*)storage[u])->print();  if (u == y)  {  std::cout << '\n';  if (y == 0)  y = 2;  else  y += y \* 2;  }  probel /= 2;  }  std::cout << '\n';  }  void Heap::minHeapify(int ix)  {  int smallest = ix;  int l = left(ix), r = right(ix);  if (l != -1 && isLess(storage[l], storage[smallest]))  smallest = l;  if (r != -1 && isLess(storage[r], storage[smallest]))  smallest = r;  if (smallest != ix)  {  swap(ix, smallest);  minHeapify(smallest);  }  }  void\* Heap::extractMin()  {  if (isEmpty())  return nullptr;  int minIdx = 0;  for (int i = 1; i < size; ++i) {  if (isLess(storage[i], storage[minIdx]))  minIdx = i;  }  void\* rc = extractI(minIdx);  return rc;  }  void\* Heap::extractI(int i)  {  if (i < 0 || i >= size)  return nullptr;  void\* rc = storage[i];  storage[i] = storage[--size];    heapify(i);    return rc;  }  Heap unionHeap(Heap a, Heap b)  {    Heap ans = create(a.maxSize + b.maxSize, a.compare);  ans.size = a.size + b.size;  for (int i = 0; i < a.size; ++i)  ans.storage[i] = a.storage[i];  for (int j = 0; j < b.size; ++j)  ans.storage[a.size + j] = b.storage[j];  for (int k = (ans.size - 2) / 2; k >= 0; --k)  ans.heapify(k);  return ans;  }  }  #pragma once  struct AAA  {  int x;  void print();  };  namespace heap  {  enum CMP  {  LESS = -1, EQUAL = 0, GREAT = 1  };  struct Heap  {  int size;  int maxSize;  void\*\* storage; // данные  CMP(\*compare)(void\*, void\*);  Heap(int maxsize, CMP(\*f)(void\*, void\*))  {  size = 0;  storage = new void\* [maxSize = maxsize];  compare = f;  };  int left(int ix);  int right(int ix);  int parent(int ix);  bool isFull() const  {  return (size >= maxSize);  };  bool isEmpty() const  {  return (size <= 0);  };  bool isLess(void\* x1, void\* x2) const  {  return compare(x1, x2) == LESS;  };  bool isGreat(void\* x1, void\* x2) const  {  return compare(x1, x2) == GREAT;  };  bool isEqual(void\* x1, void\* x2) const  {  return compare(x1, x2) == EQUAL;  };  void swap(int i, int j);  void heapify(int ix);  void insert(void\* x);  void\* extractMax();  void scan(int i) const;  void minHeapify(int ix);  void\* extractMin();  void\* extractI(int i);  };  Heap create(int maxsize, CMP(\*f)(void\*, void\*));  Heap unionHeap(Heap a, Heap b);  };  #include "Heap.h"  #include <iostream>  #include <ctime>  #include <locale>  heap::CMP cmpAAA(void\* a1, void\* a2) //Функция сравнения  {  #define A1 ((AAA\*)a1)  #define A2 ((AAA\*)a2)  heap::CMP  rc = heap::EQUAL;ц  if (A1->x > A2->x)  rc = heap::GREAT;  else  if (A2->x > A1->x)  rc = heap::LESS;  return rc;  #undef A2  #undef A1  }  int main()  {  setlocale(LC\_ALL, "rus");  int choice;  heap::Heap mainHeap = heap::create(30, cmpAAA); // Основная куча  heap::Heap secondHeap = heap::create(30, cmpAAA); // Вторая куча для объединения  // Инициализация основной кучи примерами  for (int i = 0; i < 5; i++)  {  AAA\* a = new AAA;  a->x = rand() % 100;  mainHeap.insert(a);  }  do  {  std::cout << "\n1. Добавить элемент в основную кучу";  std::cout << "\n2. Извлечь максимальный элемент";  std::cout << "\n3. Вывести основную кучу";  std::cout << "\n4. Извлечь минимальный элемент";  std::cout << "\n5. Извлечь элемент по индексу";  std::cout << "\n6. Заполнить вторую кучу для объединения";  std::cout << "\n7. Объединить кучи";  std::cout << "\n8. Выход";  std::cout << "\nВаш выбор: ";  std::cin >> choice;  switch (choice)  {  case 1: // Добавление элемента  {  AAA\* a = new AAA;  std::cout << "Введите целое число: ";  std::cin >> a->x;  mainHeap.insert(a);  break;  }  case 2: // Извлечение максимума  {  AAA\* a = (AAA\*)mainHeap.extractMax();  if (a)  {  std::cout << "Извлечен максимум: " << a->x << std::endl;  delete a;  }  else  std::cout << "Куча пуста!" << std::endl;  break;  }  case 3: // Вывод кучи  mainHeap.scan(0);  break;  case 4: // Извлечение минимума  {  AAA\* a = (AAA\*)mainHeap.extractMin();  if (a)  {  std::cout << "Извлечен минимум: " << a->x << std::endl;  delete a;  }  else  std::cout << "Куча пуста!" << std::endl;  break;  }  case 5: // Извлечение по индексу  {  int idx;  std::cout << "Введите индекс: ";  std::cin >> idx;  if (idx >= 0 && idx < mainHeap.size)  {  AAA\* a = (AAA\*)mainHeap.extractI(idx);  if (a)  {  std::cout << "Извлечен элемент " << a->x << std::endl;  delete a;  }  }  else  std::cout << "Неверный индекс!" << std::endl;  break;  }  case 6: // Заполнение второй кучи  for (int i = 0; i < 5; i++)  {  AAA\* a = new AAA;  a->x = rand() % 100;  secondHeap.insert(a);  }  std::cout << "Вторая куча создана!" << std::endl;  secondHeap.scan(0);  break;  case 7:{ // Объединение куч  mainHeap = heap::unionHeap(mainHeap, secondHeap);  std::cout << "Объединенная куча:" << std::endl;  mainHeap.scan(0);  break;  }  case 8: // Выход  exit(0);  default:  std::cout << "Ошибка ввода!" << std::endl;  }  } while (true);  return 0;  } |