**Вопросы по первому модулю**

Вопросы по алгоритмам:

1. Что означают записи “f(n) = ϴ(g(n))”, “f(n) = O(g(n))” и “f(n) = Ω(g(n))”?
2. Чем плох рекурсивный алгоритм вычисления n-ого числа Фибоначчи?
3. Опишите алгоритм проверки числа на простоту за O(sqrt(n))?
4. Опишите алгоритм возведения действительного числа в натуральную степень n за O(log n)?
5. Опишите нерекурсивный алгоритм бинарного поиска первого вхождения элемента в массиве.
6. Какова амортизированная стоимость операции Add в реализации динамического массива с удвоением буфера? Можно ли увеличивать буфер в 1.5 раза? Как это скажется на оценке?
7. Сколько времени работает линейный поиск в односвязном списке в худшем и в лучшем случае? Сколько времени работает добавление и удаление элемента в середине списка.
8. Назовите преимущества и недостатки реализации очереди с помощью динамического массива.
9. Назовите преимущества и недостатки реализации стека с помощью односвязного списка.
10. Назовите преимущества и недостатки реализации дека с помощью динамического массива.
11. Опишите методы кучи SiftUp и SiftDown. Сколько времени они работают?
12. Опишите метод кучи BuildHeap. Сколько времени он работает?
13. Опишите метод добавления элемента в кучу. Сколько времени он работает?
14. Опишите метод извлечения максимального элемента из кучи. Сколько времени он работает?
15. Опишите АТД “Очередь с приоритетом”. Как может быть реализован этот интерфейс?

Примерные вопросы по C++:

1. Напишите функцию, выводящую все нечетные числа от 1 до n включительно.
2. Напишите функцию, получающую на вход массив целых чисел и возвращающую массив, содержащий четные числа исходного массива.
3. Напишите структуры, описывающие элемент двусвязного списка и сам список. Объявите и реализуйте метод добавления элемента в конец списка.

#include <cassert>

struct node {

node\* \_next;

node\* \_prev;

int value;

};

class double\_linked\_list {

public:

double\_linked\_list();

void push\_back(int value);

private:

node\* \_head;

node\* \_tail;

int \_size;

};

void double\_linked\_list::push\_back(int val) {

node\* new\_node = new node;

new\_node->value = val;

new\_node->\_prev = \_tail;

new\_node->\_next = nullptr;

if (\_tail == nullptr) {

assert(\_head == nullptr);

\_tail = new\_node;

\_head = new\_node;

}

else {

\_tail->\_next = new\_node;

\_tail = new\_node;

}

\_size++;

}

double\_linked\_list::double\_linked\_list() {

\_head = nullptr;

\_tail = nullptr;

\_size = 0;

}

1. Напишите структуру, описывающую элемент односвязного списка. Реализуйте функцию, разворачивающую список.

#include <cassert>

#include <iostream>

struct node {

node\* \_next;

int value;

};

class linked\_list {

public:

linked\_list();

void push(int value);

void inverse\_list();

void insert\_after(node\* element, int value);

void print() {

node\* current = \_head;

while (current != nullptr) {

std::cout << current->value << std::endl;

current = current->\_next;

}

}

private:

node\* \_head;

int \_size;

};

void linked\_list::push(int val) {

node\* new\_node = new node;

new\_node->value = val;

new\_node->\_next = nullptr;

if (\_head == nullptr) {

\_head = new\_node;

}

else {

new\_node->\_next = \_head;

\_head = new\_node;

}

\_size++;

}

linked\_list::linked\_list() {

\_head = nullptr;

\_size = 0;

}

void linked\_list::inverse\_list() {

if (\_head == nullptr) {

return;

}

node \*current = \_head;

while (current->\_next != nullptr) {

current = current->\_next;

}

node \*last = current;

current = \_head;

while (current != last) {

insert\_after(last, current->value);

node\* old\_node = current;

current = current->\_next;

delete old\_node;

}

\_head = last;

}

void linked\_list::insert\_after(node\* element, int value) {

node\* new\_node = new node;

new\_node->value = value;

new\_node->\_next = element->\_next;

element->\_next = new\_node;

}

void main() {

linked\_list l;

l.push(1);

l.push(2);

l.push(3);

l.push(4);

l.push(5);

l.inverse\_list();

l.print();

system("pause");

}