C++

Урок 23

Повтороение

Приведите примеры битовых операций?

Где применяются?

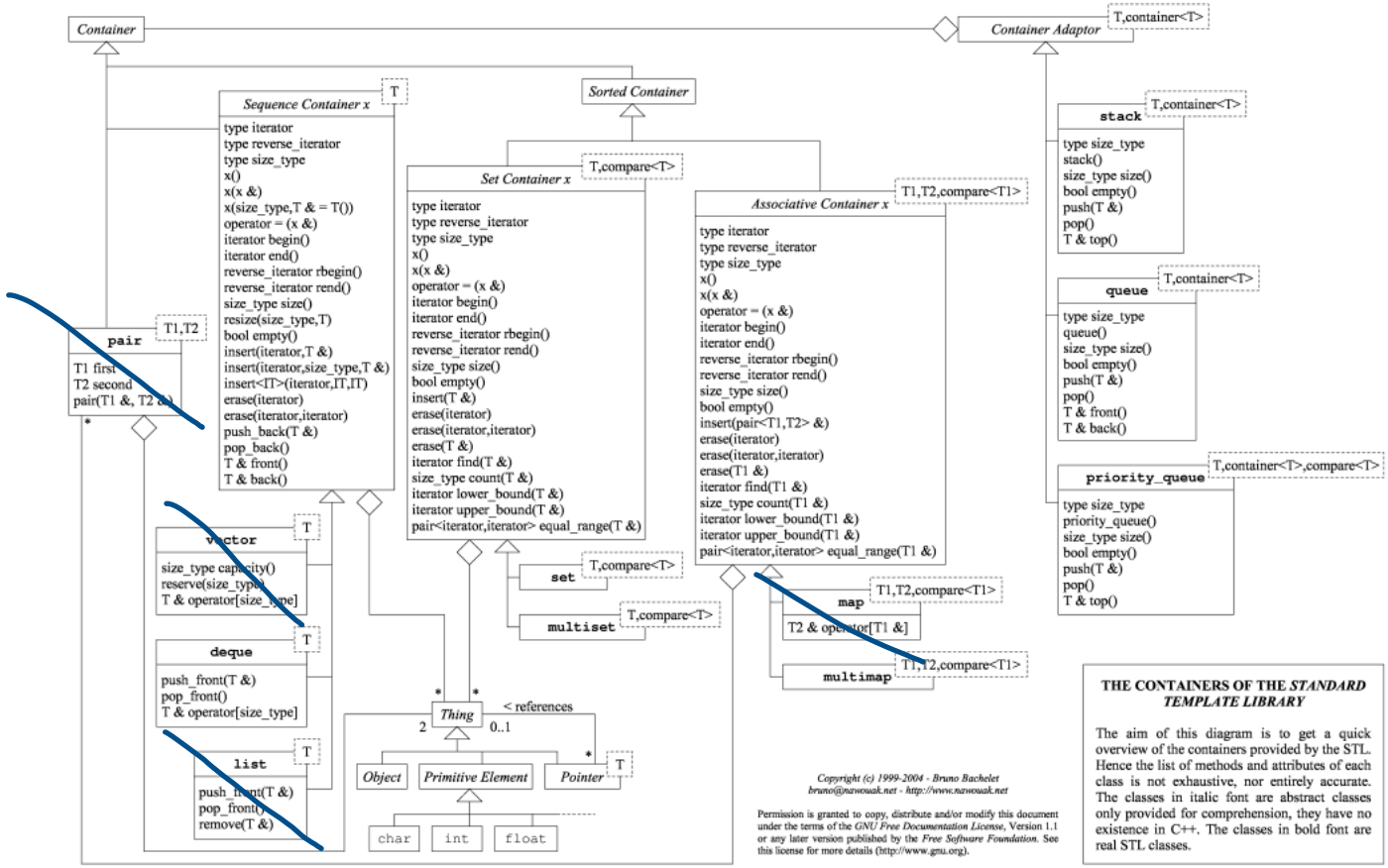
Что значит сдвиг влево на 2?

Что значит сдвиг вправо на 3?

Контейнеры 3 часть

STL

Контейнеры есть стандартные структуры данных, такие как список (**list**), вектор (**vector**), словарь (**map**) и многие другие.



Дэк

Стек

Дек (deque — double ended queue, «двусторонняя очередь») – структура данных типа «список + массив», функционирующая одновременно по двум принцам организации данных: FIFO и LIFO.

Определить дек можно как очередь с двумя сторонами, так и стек, имеющий два конца

```cpp
std::deque<int> deque1;                  // пустая очередь
std::deque<int> deque2(5);               // deque2 состоит из 5 чисел, каждый элемент имеет значение по умолчанию
std::deque<int> deque(5, 2);             // deque3 состоит из 5 чисел, каждое число равно 2
std::deque<int> deque4{1, 2, 4, 5};      // deque4 состоит из чисел 1, 2, 4, 5
std::deque<int> deque5 = {1, 2, 3, 5};   // deque5 состоит из чисел 1, 2, 3, 5
std::deque<int> deque6({1, 2, 3, 4, 5}); // deque6  состоит из чисел 1, 2, 3, 4, 5
std::deque<int> deque7(deque4);          // deque7 – копия очереди deque4
std::deque<int> deque8 = deque7;         // deque8 – копия очереди deque7
```

инициализация

```cpp
#include <iostream>
#include <deque>

int main()
{
    std::deque<int> numbers = {1, 2, 3, 4, 5};

    int first = numbers.front();                                    // 1
    int last = numbers.back();                                      // 5
    int second = numbers[1];                                        // 2
    int third = numbers.at(2);                                      // 3
    std::cout << first << second << third << last << std::endl; // 1235

    return 0;
}
```

Обращение к элементам

```cpp
#include <iostream>
#include <deque>
#include <stdexcept>

int main()
{
    std::deque<int> numbers = {1, 2, 3, 4, 5};
    try
    {
        int n = numbers.at(7);
    }
    catch (std::out_of_range e)
    {
        std::cout << "Incorrect index" << std::endl;
    }

    return 0;
}
```

Особенность at

```cpp
#include <iostream>
#include <deque>

int main()
{
    std::deque<int> numbers = {1, 2, 3, 4, 5};

    for (int n : numbers)
        std::cout << n << "\t";
    std::cout << std::endl;

    for (int i = 0; i < numbers.size(); i++)
        std::cout << numbers[i] << "\t";
    std::cout << std::endl;

    for (auto iter = numbers.begin(); iter != numbers.end(); iter++)
        std::cout << *iter << "\t";
    std::cout << std::endl;

    return 0;
}
```

Перебор контейнера

```cpp
std::deque<int> numbers = {1, 2, 3, 4, 5};
numbers.push_back(6);      // { 1, 2, 3, 4, 5, 6 }
numbers.push_front(0);     // { 0, 1, 2, 3, 4, 5, 6 }
numbers.emplace_back(7);   // { 0, 1, 2, 3, 4, 5, 6, 7 }
numbers.emplace_front(-1); // { -1, 0, 1, 2, 3, 4, 5, 6, 7 }

std::deque<int> numbers1 = {1, 2, 3, 4, 5};
auto iter1 = numbers1.cbegin(); // итератор указывает на второй элемент
numbers1.insert(iter1 + 2, 8);  // добавляем после второго элемента
// numbers1 = { 1, 2, 8, 3, 4, 5};

std::deque<int> numbers2 = {1, 2, 3, 4, 5};
auto iter2 = numbers2.cbegin(); // итератор указывает на первый элемент
numbers2.insert(iter2, 3, 4);   // добавляем вначало три четверки
// numbers2 = { 4, 4, 4, 1, 2, 3, 4, 5};

std::list<int> values = {10, 20, 30, 40, 50};
std::deque<int> numbers3 = {1, 2, 3, 4, 5};
auto iter3 = numbers3.cbegin(); // итератор указывает на первый элемент
// добавляем в начало все элементы из values
numbers3.insert(iter3, values.begin(), values.end());
// numbers3 = { 10, 20, 30, 40, 50, 1, 2, 3, 4, 5};

std::deque<int> numbers4 = {1, 2, 3, 4, 5};
auto iter4 = numbers4.cend(); // итератор указывает на позицию за последним элементом
// добавляем после последнего элемента список { 21, 22, 23 }
numbers4.insert(iter4, {21, 22, 23});
// numbers4 = { 1, 2, 3, 4, 5, 21, 22, 23};

std::deque<int> numbers = {1, 2, 3, 4, 5};
numbers.pop_front(); // numbers = { 2, 3, 4, 5 }
numbers.pop_back();  // numbers = { 2, 3, 4 }
numbers.clear();     // numbers ={}

numbers = {1, 2, 3, 4, 5};
auto iter = numbers.cbegin(); // указатель на первый элемент
numbers.erase(iter);          // удаляем первый элемент
// numbers = { 2, 4, 5, 6 }

numbers = {1, 2, 3, 4, 5};
auto begin = numbers.begin();  // указатель на первый элемент
auto end = numbers.end();      // указатель на последний элемент
numbers.erase(++begin, --end); // удаляем со второго элемента до последнего
// numbers = {1, 5}
```

Остальные ф-ии

Адаптеры

Адаптеры

Адаптеры - шаблонные классы, которые обеспечивают отображения интерфейса.

Адаптеры могут реализовываться на основе контейнера-шаблона, но в зависимости от вида ограничивать (скрывать) доступ к его методам для удобства работы с данными.

Стек

Стек

Стек - структура данных, которая построена на **односвязном списке**.

Принцип **LIFO** = Last In First Out, «последним пришел, первым вышел».

Любой элемент знает только о **следующем** элементе.

Стек

Особенность:

- нет итераторов;
- нет индексов, как в массиве (односвязный список);
- добавлении и удалении элементов выполняется за константное время;

Задача

Задача: На прием к ветеринару пришли владельцы с животными.

Занесите вес животных в **стек**. Замените все отрицательные значения на 0, а затем выведите все.

```cpp
#include <iostream>
#include <stack>

using namespace std;

int main()
{
    stack<double> weights;
    int col;
    cout << "Введите кол-во животных" << endl;
    cin >> col;
    while (col--)
    {
        double weight;
        cout << "Введите вес животного" << endl;
        cin >> weight;
        if (weight < 0)
            weight = 0;
        weights.push(weight);
    }

    cout << "Животных на приеме: " << weights.size() << endl;

    while (!weights.empty())
    {
        cout << weights.top() << " ";
        weights.pop();
    }
}
```

Код задачи

Очередь

очередь

Очередь - структура данных, которая может быть построена на основе vector, list, deque.

Принцип **FIFO** = First In First Out, «первым пришел, первым вышел».

Мы знаем только первый и последний элемент.

очередь

Особенность:

- нет итераторов;
- нет индексов, как в массиве;
- Добавление происходит в начало, а удаление с конца;
- По умолчанию основывается на deque;
- Не может быть создан на основе вектора;
- Нет возможности добавлять/удалять в центре.

очередь

```
queue<int, list<int>> val1;   // очередь на основе списка
queue<int, vector<int>> val2; // очередь на очнове вектора
```

Задача

Задача: На прием к ветеринару пришли владельцы с животными.

Занесите вес животных в **очередь**. Замените все отрицательные значения на 0, а затем выведите все.

```cpp
#include <iostream>
#include <stack>

using namespace std;

int main()
{
    stack<double> weights;
    int col;
    cout << "Введите кол-во животных" << endl;
    cin >> col;
    while (col--)
    {
        double weight;
        cout << "Введите вес животного" << endl;
        cin >> weight;
        if (weight < 0)
            weight = 0;
        weights.push(weight);
    }

    cout << "Животных на приеме: " << weights.size() << endl;

    while (!weights.empty())
    {
        cout << weights.top() << " ";
        weights.pop();
    }
}
```

Код задачи

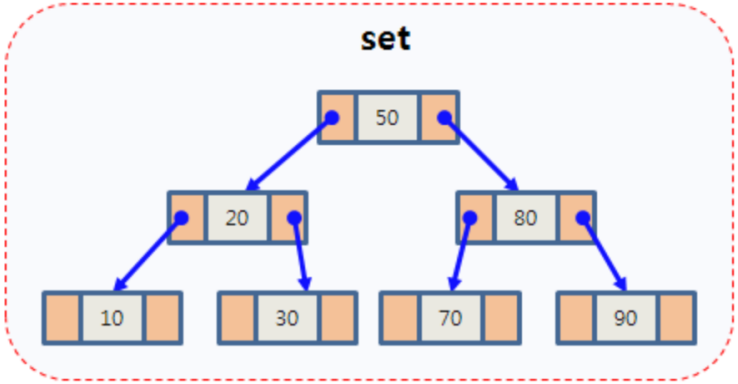
Set

Множество

set — это контейнер, который автоматически сортирует добавляемые элементы в порядке возрастания.

При добавлении одинаковых значений, set будет хранить только **один** его экземпляр.

По другому его еще называют **множеством**.



Множество

Применение

сортирует добавляемые

Multiset

Мультимножество

multiset — это контейнер, который также будет содержать элементы в отсортированном порядке при добавлении, но он хранит **повторяющееся** элементы, по сравнению с множеством set.

Часто его называют **мультимножество**.