Информатика: семестр 3

Программирование на C++

Конспекты лекций

Лектор: Хирьянов Т.Ф.

2 января 2018 г.

Оглавление

Лекция 7

Контейнеры в C++

7.1 LinkedList

Продолжим работу над кодом из предыдущей лекции:

Код 7.1: LinkedList

```cpp
#include <iostream>
#include <string>
#include "linkedList.hpp"

template<typename DataType>
struct tNode
{
    DataType data;
    tNode* next = nullptr;
};

template<typename DataType>
tNode<DataType> * insert_node(tNode<DataType> *p_begin, DataType data)
{
    tNode<DataType> *p = new tNode<DataType>;
    p->data = data;
    p->next = p_begin;
    return p;
}

int main()
{
    tNode<std::string> *p_begin = nullptr;

    p_begin = insert_node(p_begin, std::string("Hello"));
    p_begin = insert_node(p_begin, std::string("World"));
    return 0;
}
```

Код 7.2: Header-файл для нашей программы

```
// Added module

template<typename DataType>
struct tNode
{
    DataType data;
    tNode* next;
};

template<typename DataType>
struct List{
    tNode<DataType> *begin;
}

template<typename DataType>
void list_init(List<T>);

template<typename DataType>
void list_insert(List<T>, const T &value);

template<typename DataType>
void list_print(List<T>);

template<typename DataType>
void list_destroy(List<T>);

template<typename DataType>
tNode<DataType> * insert_node(tNode<DataType> *p_begin, DataType data);
```

Код заполнен не до конца, я не смог поспевать за Т.Ф. (если у кого будут исправления, прошу написать мне).

ОБНОВЛЕНИЕ. Код всего проекта можно смотреть здесь.

7.2 Библиотека стандартных шаблонов C++

Библиотека стандартных шаблонов C++ называется **STL**. Она состоит из нескольких частей и типов, но выдерживает следующую структуру:

1. Контейнеры – некоторый объект, содержащий данные.
2. Итераторы – позволяют *итерироваться по элементам контейнера.*
3. Алгоритмы – название само говорит о предназначении. Примечательно, что они не имеют привязки к конкретному типу контейнера.

Таблица (для наглядности)

Контейнеры	Итераторы	Алгоритмы
Типы контейнеров: 1. **vector<T>** (похож на **list** в Питоне) 2. **list<T>** (это **LinkedList** из Питона) 3. **set<T>** – красно-черное двоичное дерево поиска ($O(\log N)$ - добавление/удаление/поиск. 4. **unordered_set<T>** – хэш-таблица $O(1)$ – добавление/удаление/поиск 5. **map<T, V>** – ассоциативный массив, состоящий из пар *<key, value>*. При этом *key* типа **T**, *value* типа **V**	Типы итераторов: 1. input output – доступ к элементу содержимого контейнера по итератору 2. bidirectional −−i – если контейнер 'знает', что он двунаправленный 3. random access i = i + 100 – итератор произвольного доступа (а не только по +1) 4. forward ++i – обычный шаг на один вперед	1. см. ниже

Итерирование

Попробуем проитерироваться по контейнеру и применить к каждому элементу функцию **foobar**. Это можно сделать несколькими способами:

1.

Код 7.3: Пример итерирования

```
vector<int> A = {10, 3, 17, 28};

for(vector<int>::iterator i = A.begin(); i != A.end(); ++i)
{
    foobar(*i);
}
```

Здесь итератор **i** пробегает по всем значениям массива. Как он ведет учет текущей позиции, нас не волнует – в этом и есть прелесть подобной структуры. Начало итерирования – начало массива, конец итерирования – конец массива. Есть и защита: если попытаться выйти за A.end() (например, на единицу), то при попытке разыменовать такой адрес выйдет ошибка.

2. То же самое итерирование можно было провести с помощью **foreach** (из библиотеки **<algorithm>**)

Код 7.4: Применение алгоритмов для итерирования

```
#include <algorithm>

int main()
{
    vector<int> A = {10, 9, 15, 7, 20};
    for_each(A.begin(), A.end(), foobar);
}
```

Обратите внимание: функции **for_each** уходит *указатель* на функцию **foobar** (она ведь тоже хранится в памяти, как и переменные).

В стандарте C++ 11 использование контейнеров упрощено:

Код 7.5: Контейнеры в C++11

```
#include <algorithm>

int main()
{
    vector<int> A = {10, 9, 15, 7, 20};
    //for_each(A.begin(), A.end(), foobar);
    for(auto x : A) {
        x = x*2
        std::cout << x;
    }
}
```

Единственный минус: чтобы понять, что происходит в итераторе, приходится копать код вверх, до описания функции **foobar**.
В примере 7.5 используется **auto** – автоматический выбор типа. При таком подходе компилятор сам подбирает подходящий тип для переменной.