

STL: Smart pointers and containers (continuation)

To ease memory management in C++, a smart pointer has been introduced: https: //en.wikipedia.org/wiki/Smart pointer

Practice

You are asked to implement the Person class, which can add and list its friends.

The system architect decided that the Person shall have the following attributes and methods:

```
#include <memory>
#include <iostream>
#include <map>
using namespace std;
class Person
    string name;
    const int uuid; // for simplicity an int is used instead boost::uuid
    map< int, weak_ptr<Person> > friends;
public:
    explicit Person(string _name); // for simplicity use rand() instead
    ~Person();
    void AddFriend(shared_ptr<Person> &p);
    void PrintFriends();
};
```

Sample usage:

```
#include "Person.h"
int main()
    srand(0); // initialize random seed
```

```
auto katie = make_shared<Person>("Katie");
auto john = make_shared<Person>("John");
auto alice = make_shared<Person>("Alice");
auto bob = make shared<Person>("Bob");
auto another_person_called_bob = std::make_shared<Person>("Bob");
katie->AddFriend(alice);
katie->AddFriend(bob);
john->AddFriend(bob);
john->AddFriend(bob); // try to insert duplicate object
john->AddFriend(another_person_called_bob); // try to insert duplicate no
katie->AddFriend(john); // create a cyclic reference
john->AddFriend(katie); // create a cyclic reference
cout << "----" << endl:
katie->PrintFriends();
john->PrintFriends();
cout << "----" << endl:
return EXIT_SUCCESS;
```

Output:

```
Katie (uuid:1804289383) created
John (uuid:846930886) created
Alice (uuid:1681692777) created
Bob (uuid:1714636915) created
Bob (uuid:1957747793) created
Person with uuid 1714636915 and name Bob is already a friend of John
______
Katie have following friends:
   John (uuid:846930886)
   Alice (uuid:1681692777)
   Bob (uuid:1714636915)
John have following friends:
   Bob (uuid:1714636915)
```

```
Katie (uuid:1804289383)
    Bob (uuid:1957747793)
Bob (uuid:1957747793) died
Bob (uuid:1714636915) died
Alice (uuid:1681692777) died
John (uuid:846930886) died
Katie (uuid:1804289383) died
Process finished with exit code 0
```

More practise

The objects have been properly disposed. What about pointers? Consider following scenario:

```
auto thomas = make_shared<Person>("Thomas");
   john->AddFriend(thomas);
   john->PrintFriends();
} // thomas died when he went out of scope
john->PrintFriends();
cout << "-----" << endl:
return EXIT_SUCCESS;
```

Naive solution

Add a ClearUnreachableFriends() method.

```
cout << "----" << endl:
```

```
auto thomas = make_shared<Person>("Thomas");
   john->AddFriend(thomas);
   john->PrintFriends();
} // thomas died when he went out of scope
john->PrintFriends();
john->ClearUnreachableFriends();
john->PrintFriends();
cout << "----" << endl:
return EXIT SUCCESS;
```

Better solution (for ambitious)

Add a container storing people_who_likes_me to notify them when the object is destructed. As a result they will be able do clean up dangling pointers.

STL ciąg dalszy - old

Zadanie 1

Utwórz obiekt Record służacy do przechowywania pewnych danych o osobach np.:

```
class Record
public:
    std::string mName;
    std::string mPhone;
    int
            mAge;
};
```

Zadanie 2

Dopisz do powyższego obiektu konstruktory domyślny Record() i jednoargumentowy Record(const char c[]) który inicjalizuje atrybut mName wartościa argumentu c.

Zadanie 3

W funkcji głównej programu zadeklaruj tablicę obiektów typu Record korzystajac ze standardowego kontenera vector.

Zadanie 4

Wypełnij tablice pewna ilościa danych tak aby kilka razy powtórzyły się osoby o tym samym nazwisku np. "Nowak" i występowały inne nazwiska zaczynające się na litere N.

Zadanie 5

Skopiuj powyższą tablicę do innej tablicy tymczasowej tego samego typu korzystając z algorytmu copy(itr_beg, itr_end, dest_itr_beg) gdzie itr_beg i itr_end określając zakres źródła do skopiowania a dest_itr_beg określa początek kontenera gdzie mają być wstawiane elementy.

Zadanie 6

Wydrukuj zawartość tablicy na ekran korzystając z algorytmu for_each(...). Ostatnim argumentem tego algorytmu jest funkcja lub obiekt funkcyjny dokonujący operacji na każdym z obiektów kolekcji znajdującym się pomiędzy podanymi iteratorami. Sprawdź działanie tego algorytmu korzystając zarówno z funkcji jak i z obiektu funkcyjnego. Przykładowy obiekt funkcyjny wygląda tak:

```
class Print
public:
            operator () (const Record& rec)
    void
    {
        // instrukcje które będą wykonywane
};
```

Zadanie 7

Wydrukuj zawartość tablicy do pliku korzystając ze strumienia ofstream i zmodyfikowanego obiektu funkcyjnego, który będzie dodatkowo przechowywał strumień (konieczny konstruktor!).

Zadanie 8

Znajdż w tablicy wszystkie wystąpienia osób z nazwiskiem "Nowak". tym celu posortuj (algorytm sort) tablice a następnie skorzystaj z algorytmu equal_range(itr_begin, itr_end, val) który zwraca obiekt typu pair<iterator, iterator> a jego atrybuty first i second tej pary odpowiednio przechowuja iteratory do pierwszego i znajdującego się za ostatnim obiektu równego val. Aby można było skorzystać z tego algorytmu niezbędne jest dopisanie operatora < do obiektu Record. Wynik wydrukuj na ekran korzystając z for_each.

Zadanie 9

Znajdź wszystkie elementy tablicy dla których nazwisko osoby zaczyna się na literę N. Skorzystaj z algorytmu lower_bound(itr_begin, itr_end, val) który zwraca iterator do pierwszego elementu o wartości nie mniejszej niż val.