БИБЛИОТЕКА СТАНДАРДНИХ ШАБЛОНА: МАПА И СКУП

STL Map Class – шаблонска класа мапе

Под појмом мапа, овде се подразумева асоцијативни низ, то јест, ради се о пресликавању (енг. mapping, па отуд и назив „мапа“) једног скупа елемената у други.

Претпоставимо да радимо са подацима који за сваку вредност имају придружен знаковни низ, односно стринг. На пример, уз свако име студента придружимо његову оцену, или број индекса. Како бисмо овакву структуру описали језиком Це++? Један начин би био да самостално израдимо неку врсту табеле (или пак хеш, енг. *hash*, табеле). Тај приступ подразумева и писање сопствене функције за претраживање табеле, бригу о граничним ситуацијама као и много испитивања како би били сигурни да све ради како треба. Са друге стране, библиотека стандардних шаблона већ садржи класу која одговара управо овом случају: класу *map*.

Питање: Шта је хеш табела?

Мапе нам омогућавају да асоцирамо било која два типа података. За дефиницију променљиве типа класе мапе, потребно је задати два, односно три типа података:

```
std::map<key_type, data_type, [comparison_function]>
```

Уочавамо да је функција поређења (***comparison_function***) у угластим заградама, што значи да овај параметар није обавезан докле год класа која дефинише кључ (***key_type***) има дефинисан оператор „мање од“, тј. <. Оператор < обезбеђује да елементи у мапи буду поређани по реду. То значи да ће и приступ елементима помоћу итератора бити по редоследу кључева.

У следећем примеру показано је чување података о студентским оценама:

```
std::map<std::string, int> grade_list;
```

Сада се за приступ студентским оценама структура може третирати као низ и користити оператор []. Једина разлика је што се сада за индекс низа користи тип кључа при дефиницији структуре.

Придруживање оцене 9 студенту „Djura“ ради се на следећи начин:

```
grade_list["Djura"] = 9;
```

Ако је у међувремену студент поправио оцену, измена се врши на идентичан начин:

```
grade_list["Djura"] = 9;
// Ђура се поправио
grade_list["Djura"] = 10;
```

Са друге стране, брисање елемента се обавља позивом функције ***erase()*** која припада класи мапе.

Брисање студента „Djura" обавиће се следећом линијом:

```
grade_list.erase("Djura");
```

Следи још неколико корисних метода класе мапа. Број елемената добија се позивом функције ***size()***, која је такође припадник класе мапа. Још једна корисна функција је ***empty()*** и користи се за проверу да ли је структура мапе празна. Брисање свих елемената обавља се позивом функције ***clear()***. Приметите да и листа и вектор имају ове методе.

Уколико је потребно проверити присутност неког елемента у структури, употреба оператора [] није препоручљива, јер нисмо сигурни који податак ће бити враћен ако елемент не постоји. У том случају користи се функција ***find()*** која враћа итератор траженог елемента уколико он постоји, или исти итератор који враћа функција ***end()***. уколико тај елемент не постоји. Следећи пример показује употребу функције ***find()***:

```
std::map<std::string, int> grade_list;
grade_list["Djura"] = 10;
if (grade_list.find("Pera") == grade_list.end())
{
    std::cout << "Pera se ne nalazi u mapi!" << endl;
}
```

За приступ елементима се такође могу користити итератори као што је већ било показано. Елементу на који показује итератор класе мапа се приступа на мало другачији начин. У овом случају итератор показује на пар – кључ и упарену вредност. Овим елементима итератора се приступа преко променљивих ***first*** и ***second***. Пошто се итератор третира као показивач, овим елементима се приступа преко оператора ->, што се види из следећег примера:

```
std::map<std::string, int> grade_list;
grade_list["Djura"] = 10;
// Исписаће "Djura"
std::cout << grade_list.begin()->first << endl;
// Исписаће "10"
std::cout << grade_list.begin()->second << endl;
```

Када је у питању брзина употребе класе мапа, треба имати на уму да је трајање уноса новог елемента у структуру или претрага постојећег сразмерна log(n), где је n тренутни број елемената у структури. Овакав приступ је потенцијално знатно спорији од употребе еквивалентне хеш табеле са брзом хеш функцијом. Међутим, за разлику од хеш табле елементи у мапи су сортирани по кључу.

STL Set Class – шаблонска класа скупа.

Шаблонска класа *set* представља асоцијативни сортирани скуп у којем податак може да се појави само једном - дупликати нису дозвољени. Назива се асоцијативном структуром зато што се за потребе сортирања користи сам податак.

За дефиницију промењиве типа класе скуп потребно је дефинисати тип података који ће бити смештен унутра:

```
template<class Key, [comparison_function]> class set {...};
```

Као и услучају класе мапа, у угластим заградама дефинише се функција поређења (***comparison_function***) која представља необавезни параметар. Ако класа не поседује дефинисан оператор < потребно је дефинисати га. Уколико другачије није дефинисано подразумевани критеријум за сортирање је оператор <.

Приликом коришћења класе скуп треба имати на уму да, за разлику од класе мапе, непосредна промена вредности елемената унутар скупа коришћењем итератора није могућа из разлога што би свака измена вредности постојећег елемента потенцијално утицала на њихов поредак унутар самог скупа. Из тог разлога измена елемента обавља се тако што се елемент прво избрише из скупа, затим се измени, и на крају се поново дода у скуп. Следи пример коришћења:

```
class StudentGrade {
public:
	string student;
	int grade;
	StudentGrade(string s, int g) {
		student = s;
		grade = g;
	}
};

typedef std::set<StudentGrade>::iterator SGIter;

bool updateGrade(set<StudentGrade> &s, StudentGrade &e, int g) {
	bool retVal;
	SGIter result = s.find(e); // Пронађи елемент.

	//Да ли је елемент постоји?
	result == s.end() ? retVal = false : retVal = true;

	if (retVal) {
		//result->grade = s; // Грешка, const итератор!!!
		s.erase(result);     // Обриши елемент.
		e.grade = g;         // Ажурирај (измени) елемент.
		s.insert(e);         // Поново додај елемент.
	}

	return retVal;
}

int main() {
	set<StudentGrade> studentList;   //Направи скуп.
	StudentGrade st1("Djura", 6);
	StudentGrade st2("Petar", 7);

	studentList.insert(st1); //Додај елемент у скуп.
	studentList.insert(st2);  //Додај елемент у скуп.

	if (!updateGrade(studentList, st1, 7))   //Промени оцену студенту Djura
		cout << "Warning: Element not found" << endl;

	return 0;
}
```