Zadanie 11 26-29 maja 2025 r.

kurs języka C++

szablon listy jednokierunkowej

Instytut Informatyki
Uniwersytetu Wrocławskiego Paweł Rzechonek

Prolog

Szablony w C++ umożliwiają programowanie uogólnione, czyli definiowanie abstrakcyjnych algorytmów oraz struktur danych niezależnych od konkretnych typów, na których one pracują. Typ danych dla szablonu jest określany (w sposób jawny bądź niejawny) dopiero w miejscu użycia szablonu – to wtedy kompilator wygeneruje odpowiednią definicję funkcji albo klasy szablonowej z określonym już typem.

Zadanie

Zdefiniuj szablon klasy `mylist<T>` dla listy jednokierunkowej w przestrzeni nazw `adt`. Klasa reprezentująca listę ma być napisana zgodnie ze sztuką programowania dynamicznych struktur danych – w pełni funkcjonalny węzeł listy `mynode<T>` zdefiniuj jako prywatną klasę zagnieżdżoną w klasie `mylist<T>`; klasa listy `mylist<T>` będzie więc wygodnym do używania opakowaniem na ukrytą homogeniczną strukturę listową zbudowaną na węzłach `mynode<T>`. W szablonie klasy `mylist<T>` zdefiniuj następującą funkcjonalność (analogiczną funkcjonalność zdefiniuj też w klasie węzła `mynode<T>`):

a. wstawienie elementu na początek listy (czyli na pozycję 0);
b. wstawienie elementu na koniec listy;
c. wstawienie elementu na zadaną pozycję;
d. usunięcie elementu z początku listy (czyli z pozycji 0);
e. usunięcie elementu z końca listy;
f. usunięcie elementu z określonej pozycji;
g. usunięcie elementu o zadanej wartości (pierwszego od początku);
h. usunięcie wszystkich elementów o zadanej wartości;
i. określenie pozycji elementu o zadanej wartości (pierwszego od początku);
j. policzenie wszystkich elementów o zadanej wartości;
k. zliczenie wszystkich elementów na liście;
l. sprawdzenie czy lista jest pusta.

Obiekt listy `mylist<T>` ma być kopiowalny (konstruktor oraz przypisanie kopiujące i przenoszące). Uzupełnij definicję szablonu o konstruktor, który zainicjalizuje listę wartościami początkowymi przekazanymi za pomocą kontenera `initializer_list<T>`. Pamiętaj również o operatorze strumieniowym `operator<<` do czytelnego wypisania zawartości listy.

Następnie zdefiniuj szablony klas implementujących kolejkę myqueue<T> i stos mystack<T>. Szablony te mają być zbudowane na liście (dziedziczenie niepublicznie po mylist<T>). Próba pobrania elementu z pustej kolejki albo z pustego stosu ma skutkować zgłoszeniem jakiegoś wyjątku standardowego.

Dalej, w przestrzeni nazw adt, zdefiniuj szablony dwóch funkcji do pracy z listami:

a. funkcja issorted() ma sprawdzać, czy lista jest uporządkowana;
b. funkcja sort() ma posortować elementy na liście.

Szablony tych funkcji powinny posiadać dwa parametry: typ danych przechowywanych w liście oraz trejta implementującego operację porównywania elementów wybranego typu. Trejt ma być parametrem domyślnym w szablonie ustawionym na obiekt lessthan<T> zawierający operację porównywania za pomocą zwykłego operatora <; zdefiniuj też komplementarnego trejta greaterthan<T> implementującego porównywanie za pomocą operatora >. W trejtach zadbaj o specjalizację dla wskaźników na obiekty a w szczególności dla wskaźnika typu const char*.

Na koniec w funkcji main() napisz zestaw testów rzetelnie sprawdzających działanie wszystkich zdefiniowanych klas i funkcji, pracujących na różnych typach danych. Obiekty list, stosów i kolejek, które będą poddawane testowaniu stwórz na stercie operatorem new; nie zapomnij zlikwidować ich operatorem delete przed zakończeniem programu!

Istotne elementy programu.

- Podział programu na pliki nagłówkowe i pliki źródłowe (osobny plik z funkcją main()).
- Użycie przestrzeni nazw adt.
- Definicja szablonu klasy dla listy jednokierunkowej mylist<T> wraz z zagnieżdżoną definicja węzła mynode<T>.
- Szablony klas dla kolejki myqueue<T> i stosu mystack<T>.
- Zgłaszanie wyjątków w stosie i kolejce przy próbie pracy z pustą kolekcją.
- Szablony funkcji do sortowania danych na liście sort() i do weryfikacji posortowania issorted().
- Definicja trejtów lessthan<T> i greaterthan<T> realizujących porównania.
- Realizacja specjalizacji trejtów dla wskaźników a w szczególności dla wskaźnika typu const char*.
- Implementacja kopiowania i przenoszenia dla listy mylist<T>.
- Inicjalizacja stanu początkowego listy za pomocą kolekcji initializer list<T>.
- Destrukcja listy.
- Podział programu na pliki nagłówkowe i pliki źródłowe (wyodrębniony osobny plik z funkcją main() z testami).