Języki programowania, 23.12.2020
Zadanie 11

1. Klasa szablonowa 1 p.

Wstęp – co to są klasy i funkcje szablonowe w języku C++?

Rozważmy prostą klasę, która przechowuje jakiś typ informacji, na przykład tablicę `int`'ów:

```
class KlasaInt
{
      int *tablica
      int rozmiar;
      public:
      KlasaInt(int Rozmiar);
      void UstawElement(int poz, int wartosc);
      int PobierzElement(int poz);
};

KlasaInt::KlasaInt(int Rozmiar)
{
      rozmiar = Rozmiar;
      tablica = new int[Rozmiar];
}
void KlasaInt::UstawElement(int poz, int wartosc)
{
      tablica[poz]=wartosc;
}
int KlasaInt::PobierzElement(int poz)
{
      return tablica[poz];
}

int main()
{
      KlasaInt kl(10);
      kl.UstawElement(5) = 20;
      cout<<kl.PobierzElement(5)<<endl;


      return 0;
}
```

Co by należało zrobić, gdybyśmy potrzebowali przechować w identyczny sposób dodatkowo inne typy danych. np. `double`, `std::string`, `char*`, etc.? Musielibyśmy za każdym razem, dla każdego oddzielnego typu danych, który jest nam potrzebny, tworzyć nową klasę (np. `KlasaString`, gdzie mielibyśmy `string *tablica` zamiast `int *tablica` itp.). Napisanie sporej ilości tego typu klas zajęłoby nam pewną ilość czasu oraz zwiększyło znacznie ilość plików w projekcie (można sobie wyobrazić duży projekt, gdzie każdy dodaje klasy różniące się właściwie tylko typem przechowywanych danych a służących do tych samych celów – ogrom klas!). Na szczęście język C++ przychodzi nam tu z rozwiązaniem takiego problemu i pozwala tworzyć klasy szablonowe (ang. *template class*), które dostosowują się automatycznie do typu danych używanego w programie głównym. Przykład klasy szablonowej analogicznej do przykładu powyżej jest następujący:

```
template<class T>
class KlasaSzablonowa
{
      T *tablica
      int rozmiar;
      public:
      KlasaSzablonowa(int Rozmiar);
      void UstawElement(int poz, T wartosc);
      T PobierzElement(int poz);
};

template<class T>
KlasaSzablonowa<T>::KlasaSzablonowa(int Rozmiar)
{
      rozmiar = Rozmiar;
      tablica = new T[Rozmiar];
}
```

```
template<class T>
void KlasaInt<T>::UstawElement(int poz, T wartosc)
{
	tablica[poz]=wartosc;
}

template<class T>
T KlasaInt<T>::PobierzElement(int poz)
{
	return tablica[poz];
}

int main()
{
	KlasaSzablonowa<int> klInt(10);
	klInt.UstawElement(5) = 20;
	cout<<klInt.PobierzElement(5)<<endl;

	KlasaSzablonowa<string> klString(10);
	klString.UstawElement(5) = "tekst";
	cout<<klString.PobierzElement(5)<<endl;

	return 0;
}
```

Załóżmy teraz, że w naszej klasie szablonowej mamy również metodę `Dodaj`, która zwraca sumę elementów dwóch obiektów z wybranych pól tablicy `tablica`, czyli:

```
template<class T>
T KlasaSzablonowa<T>::Dodaj(int poz1, int poz2)
{
	T obiekt = tablica[poz1]+tablica[poz2];
	return obiekt;
}
```

Taka metoda zadziała dla prostych typów danych, np. `int`, `double`, `string`, itp. (czyli takich, gdzie mamy zdefiniowany operator + i takie działanie ma sens). Co by się jednak stało, gdyby nasza klasa przechowywała na przykład napisane wcześniej obiekty klasy `Histogram`?(czyli coś bardziej skomplikowanego)? Taka metoda oczywiście nam nie zadziała dla histogramu (nie zdefiniowaliśmy przeciążonego operatora + dla klasy `Histogram` i jego użycie wyrzuciłoby nam błąd). Nic nie szkodzi! Język C++ przynosi nam i tutaj rozwiązanie – można tworzyć specjalizowane metody w klasach szablonowych, oto przykład metody `Dodaj` dla histogramów:

```
template<>
Histogram KlasaSzablonowa<Histogram>::Dodaj(int poz1, int poz2)
{
	Histogram c = tablica[poz1];
	for(int i=0;i<c->IloscBinow())
		c.UstawLiczbeZliczen(i,tablica[poz1]->PobierzLiczbeZliczen(i)
+tablica[poz]->PobierzLiczbeZliczen(i));
	return c;
	//to tylko przykład - tu robimy cokolwiek z obiektami typu Histogram, np.
tworzymy wyjsciowy histogram gdzie w kazdym binie liczba zliczen jest suma z
histogramow tablica[poz1] i tablica[poz2]
}
```

W naszej klasie możemy mieć wiele specjalizowanych metod (również wiele specjalizowanych metod o tej samej nazwie – dla różnych obiektów). Kompilator w trakcie kompilowania programu sam zdecyduje, którą metodę ma użyć (w przypadku wszystkich typów obiektów użyje metody ogólnej `Dodaj` z operatorem +, ale jeśli nasza klasa będzie akurat przechowywać histogramy, to automatycznie użyje metody specjalizowanej dla klasy `Histogram`).

Uwaga!

W przypadku klas szablonowych wszystko (zarówno definicję klasy jak i metody klasy) piszemy tylko w pliku **.h** (czyli najczęściej dla jednej klasy jeden plik .h) – nie możemy pisać definicji metod w pliku **.cpp**!

W języku C++ oprócz klas szablonowych możemy również tworzyć funkcje szablonowe. Np. załóżmy, że mamy kilka klas, które mają taką samą metodę (dajmy na to `int GetRozmiar()`). Chcielibyśmy stworzyć funkcję, która porównuje rozmiar dwóch obiektów dowolnej z tych klas i zwraca większą wartość. Nic trudnego, możemy dostawić więcej typów danych:

```
template<class T1, class T2>
int CompareRozmiar(T1 obiekt1, T2 obiekt2)
```

```
{
	if(obiekt1.GetRozmiar()>obiekt2.GetRozmiar())
		return obiekt1.GetRozmiar();
	else
		return obiekt2.GetRozmiar();
}
```

Oczywiście możemy dodawać więcej typów klas w nagłówku `template`, możemy je też dowolnie nazywać (np. `template<class A, class B, class C, class D>` itp.).

Chcemy rozszerzyć jej działanie napisanej na wcześniejszych zajęciach klasy `List` na elementy dowolnego typu.
Zasada działania naszej klasy z punktu widzenia programu będzie praktycznie identyczna jak listy ze standardowej biblioteki szablonów STL, np.:

```
List<double> *myList = new List<double>(); //tworzy liste przechowujaca typ double za pomoca naszej klasy List
list<double> *myListSTL = new list<double>(); //tworzy liste przechowujaca typ double za pomoca klasy z biblioteki STL
```

Uwaga!
Należy posiadać poprawnie działającą wersję klasy `List` oraz poprawnie działającą klasę z histogramem – `Histogram`.
Klasy szablonowe piszemy **tylko w plikach .h** – nie tworzymy pliku **.cpp** (ponieważ klasa szablonowa jest kompilowana dopiero przy jej użyciu w trakcie wykonywania programu).
`Używamy pliku Makefile w najbardziej zaawansowanej postaci.`