0.1 Ulepszenie programowania generycznego

Specyfikacja konceptów zawiera wiele ulepszeń, by lepiej wspierać programowanie generyczne przez:

- umożliwienie wyraźnego określania ograniczeń argumentów szablonu jako części deklaracji szablonów
- wsparcie możliwości przeciążania szablonów funkcji i częściowego określania szablonów klas i zmiennych opartych na tych ograniczeniach
- dostarczenie składni do definiowania konceptów i wymagań narzuconych na argumenty szablonu
- ujednolicenie `auto`[1] i konceptów w celu zapewnienia jednolitej i dostępnej notacji dla programowania ogólnego
- radykalną poprawę jakości wiadomości błędów wynikających z niewłaściwego wykorzystania szablonów
- osiągnięcie tego wszystkiego bez żadnego narzucania jakichkolwiek dodatkowych zasobów ani znacznego wzrostu czasu kompilacji, bez ograniczania tego, co można wyrazić przy użyciu szablonów

```
double pierwiastek(double d);
double d = 7;
double d2 = pierwiastek(d);
vector<string> v = {"jeden", "dwa"};
double d3 = pierwiastek(v);
```

Funkcja `pierwiastek`, która jako parametr przyjmuje zmienną typu `double`. Jeśli zostanie jej dostarczony taki typ, wszystko będzie w porządku, ale jeśli inny, od razu kompilator wyprodukuje pomocną wiadomość błędu.

Kod funkcji `sortuj` zależy od różnych właściwości typu `T`, takiej jak posiadanie operatora `[]`.

[1] Słowo kluczowe wg standardu *C++11*, oznaczające zastępczy typ zmiennej, który zostanie wydedukowany na podstawie wartości za pomocą której zmienna zostanie zainicjalizowana.

```
template<class T>
void sortuj(T &c){
    //kod sortowania
}
```

```
vector<string> v = {"jeden", "dwa"};
sortuj(v);
//OK: zmienna v ma wszystkie syntaktyczne właściwości wymagane
przez funkcję sort

double d = 7;
sortuj(d);
//Błąd: zmienna d nie ma operatora []
```

Pojawiło się kilka problemów:

- wiadomość błędu jest niejednoznaczna i daleko jej do precyzyjnej i pomocnej, tak jak : "Błąd: zmienna d nie ma operatora []"
- aby użyć funkcji `sortuj`, musimy dostarczyć jej definicję, a nie tylko deklaracje. Jest to różnica w sposobie pisania zwykłego kodu i zmienia się model organizowania kodu
- wymagania funkcji dotyczące typu argumentu są domniemane w ciałach ich funkcji
- wiadomość błędu funkcji pojawi się tylko podczas inicjalizacji szablonu, a to może się zdarzyć bardzo długo po momencie wywołania
- Notacja `template<typename T>` jest powtarzalna, bardzo nielubiana.

Używając konceptu, możemy dotrzeć do źródła problemu, poprzez poprawne określenie wymagań argumentów szablonu. Fragment kodu używającego konceptu `Sortable`:

```
void sortuj(Sortable &c);//(1)
vector<string> v = {"jeden", "dwa"};
sortuj(v);//(2)
double d = 7;
sortuj(d);//(3)
```

(1) - akceptuj jakąkolwiek zmienną `c`, która jest typu `Sortable`
(2) - OK: `v` jest kontenerem typu `Sortable`
(3) - Błąd: d nie jest `Sortable` (`double` nie dostarcza operatora [], itd.)

Kod jest analogiczny do przykładu `pierwiastek`. Jedyna różnica polega na tym, że:

- w przypadku typu `double`, projektant języka wbudował go do kompilatora jako określony typ, gdzie jego znaczenie zostało określone w dokumentacji
- zaś w przypadku `Sortable`, użytkownik określił co on oznacza w kodzie. Typ jest `Sortable` jeśli posiada właściwości `begin()` i `end()` dostarczające losowy dostęp do sekwencji zawierającej elementy, które mogą być porównywane używając operatora `<`

Teraz otrzymujemy bardziej jasny komunikat błędu. Jest on generowany natychmiast w momencie gdzie kompilator widzi błędne wywołanie (`sortuj(d);`)

Cele konceptów to, zrobienie:

- kodu generycznego tak prostym jak niegeneryczny
- bardziej zaawansowanego kodu generycznego tak łatwym do użycia i nie tak trudnym do pisania