0.1 Inicjalizacje i sprawdzanie

Minimalne przetwarzanie semantyczne odbywa się, gdy po raz pierwszy widzi definicję szablonu lub jego użycie. Pełne przetwarzanie semantyczne jest przesuwane na czas inicjalizacji (tuż przed czasem linkowania), na podstawie każdej instancji. Oznacza to, że założenia dotyczące argumentów szablonu nie są sprawdzane przed czasem inicjalizacji. Np.

```
string x = "testowy tekst";
kwadrat(x);
```

Bezsensowne użycie zmiennej `string` jako argumentu funkcji `kwadrat` nie jest wyłapane w momencie użycia. Dopiero w czasie inicjalizacji kompilator odkryje, że nie ma odpowiedniej deklaracji dla operatora *. To ogromny praktyczny błąd, bo inicjalizacja może być przeprowadzona przez kod napisany przez użytkownika, który nie napisał definicji funkcji `kwadrat` ani definicji `string`. Programista, który nie znał definicji funkcji `kwadrat` ani `string` miałby ogromne trudności w zrozumieniu komunikatów błędów związanych z ich interakcją (np. "illegal operand for *").

Istnienie symbolu operatora * nie jest wystarczające by zapewnić pomyślną kompilację funkcji `kwadrat`. Musi istnieć operator *, który przyjmuje argumenty odpowiednich typów i ten operator * musi być bezkonkurencyjnym dopasowaniem według zasad przeciążania C++. Dodatkowo funkcja `kwadrat` przyjmuje argumenty przez wartość i zwraca swój wynik przez wartość. Z tego wynika, że musi być możliwe skopiowanie obiektów dedukowanego typu. Potrzebny jest rygorystyczny framework do opisywania wymagań definicji szablonów na ich argumentach.

Doświadczenie podpowiada nam, że pomyślna kompilacja i linkowanie może nie gwarantować końca problemów. Udana budowa pokazuje tylko, że inicjalizacje szablonów były poprawne pod względem typów, dostając argumenty które przekazaliśmy. Co z typami argumentów szablonu i wartościami, z którymi nie próbowaliśmy użyć naszych szablonów? Definicja szablonu może zawierać przypuszczenia na temat argumentów, które przekazaliśmy ale nie zadziała dla innych, prawdopodobnie rozsądnych argumentów. Uproszczona wersja klasycznego przykładu:

```
template<typename FwdIter>
bool czyJestPalindromem(FwdIter first, FwdIter last){
    if(last <= first) return true;
    if(*first != *last) return false;
    return czyJestPalindromem(++first, --last);
}
```

Testujemy czy sekwencja wyznaczona przez parę iteratorów do jego pierwszego i ostatniego elementu, jest palindromem. Przyjmuje się, że te iteratory

są z kategorii *forward iterator*. To znaczy, że powinny wspierać co najmniej operacje takie jak: *, != i ++. Definicja funkcji `czyJestPalindromem` bada czy elementy sekwencji zmierzają z początku i końca do środka. Możemy przetestować tę funkcję używając `vector`, tablicę w stylu `C` i `string`. W każdym przypadku nasz szablon funkcji zainicjalizuje się i wykona się poprawnie. Niestety, umieszczenie tej funkcji w bibliotece byłoby dużym błędem. Nie wszystkie sekwencje wspierają `--` i $\leqslant$. Np. listy pojedyncze nie wspierają. Eksperci używają wyszukanych, regularnych technik by uniknąć takich problemów. Jednakże, fundamentalny problem jest taki, że definicja szablonu nie jest (według siebie) dobrą specyfikacją jego wymagań na jego parametry.