

1 Podstawy stosowania pakietu listings

W trywialnym przypadku program z języku C++ może składać się tylko z jednego pliku i jednej funkcji. *Funkcja* jest wydzieloną częścią programu, realizującą pewne zadanie. Kompletny program musi zawierać funkcję o nazwie *main* od której rozpoczyna się wykonanie programu. Do programu można dołączać pliki zawierające nagłówki (opis) funkcji zdefiniowanych w innych plikach lub funkcji systemowych (dyrektywa `include`).

```
#include <iostream>
```

```
int main()
{
    std::cout << "C++\n";
}
```

```
1 #include <iostream>
2
3 int main()
4 {
5     std::cout << "C++\n";
6 }
```

Komentarz w C++, to dowolnej długości tekst ograniczony znakami `/* i */` lub tekst od znaku `//` do końca linii.

```
int main() { /* Ten program nic nie robi. */ }
```

Ada należy do języków rodziny Algol/Pascal, programy napisane w Adzie są czytelne i stosunkowo łatwe do analizy.

```
with Text_IO, Ada.Integer_Text_IO;
use Text_IO, Ada.Integer_Text_IO;

procedure Silnia is
    n, s, i : Integer := 1;

begin
    Get(n);

    while i < n loop
        i := i + 1;
        s := s * i;
    end loop;

    Put("Silnia: ");
    Put(s, 0);
end;
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17

2 Definiowanie własnego języka programowania

Język opisu dynamiki (*Alvis Code Language*, w skrócie *AlvisCD*, służy do definiowania zachowania indywidualnych agentów. Występujące w nim instrukcje są wzorowane na wybranych operatorach algebry CCS, które w XCCS stosowane były w warstwie algebraicznej. W przeciwieństwie do języków modelowania CCS i XCCS, które skupiają się głównie na opisie komunikacji, marginalizując kwestie związane z manipulowaniem wartościami parametrów, język *Alvis* pozwala na wygodne modyfikacje wartości parametrów agenta, niezależne od instrukcji dotyczących komunikacji.

```
1  environment {
2    in wakeup [] (map (60000*) [1..]) durable;
3    in off [] (map (1000*) [1..]) signal;
4    out warning [0,1,2] [];
5    out brake [] [];
6  }
7
8  agent ATS {
9    loop {
10      wakeup;
11      out warning 1;
12      select {
13        alt (ready [in(off)]) {
14          in off;
15          out warning 0;
16        }
17        alt(delay 6000) {
18          out warning 2;
19          select {
20            alt (ready [in(off)]) {
21              in off;
22              out warning 0;
23            }
24            alt (delay 3000) {
25              out brake;
26              exit;
27            }
28          }
29        }
30      }
31    }
32  }
```

UWAGA: Pakiet *listings* pamięta poprzednie ustawienia, jeśli ich nie nadpiszemy, tzn. jeżeli dla poprzedniej specyfikacji ustawiliśmy wyświetlanie numerów linii po prawej stronie, a w kolejnej specyfikacji nic o liczbach nie piszemy, to nadal będą wyświetlane po prawej stronie.

3 Podpisy, odwołania i ramki

Agenty pasywne są stosowane do opisu współdzielonych zasobów. Przykładową implementację jedno-komórkowego bufora pokazano na listingu ??.

```
12 agent Buffer {  
13   i :: Int = 0;  
14   proc pop { out pop i; }  
15   proc push { in push i; }  
16 }
```

Listing 1: Przykład agenta pasywnego

Na listingu 1 ciągle rozważamy ten sam przykład, ale w ramce ;)

Listing 2: Przykład agenta pasywnego

```
1 agent Buffer {  
2   i :: Int = 0;  
3   proc pop { out pop i; }  
4   proc push { in push i; }  
5 }
```

Listing 3: Przykład agenta pasywnego

```
1 agent Buffer {  
2   i :: Int = 0;  
3   proc pop { out pop i; }  
4   proc push { in push i; }  
5 }
```

Listing 4: Przykład agenta pasywnego

```
1 agent Buffer {  
2   i :: Int = 0;  
3   proc pop { out pop i; }  
4   proc push { in push i; }  
5 }
```

Narożniki podajemy od prawego [górnego](#) począwszy [zgodnie z](#) ruchem wskazówek zegara. Należy podać dokładnie 4 litery, *t* oznacza zaokrąglony narożnik, a *f* prosty.

Można połączyć grafikę z kodem w ramach jednego środowiska *figure*, tak jak pokazano na rysunku ??.