POLITECHNIKA GDAŃSKA

WYDZIAŁ ELEKTRONIKI, TELEKOMUNIKACJI I INFORMATYKI

Wprowadzenie do języka Ada

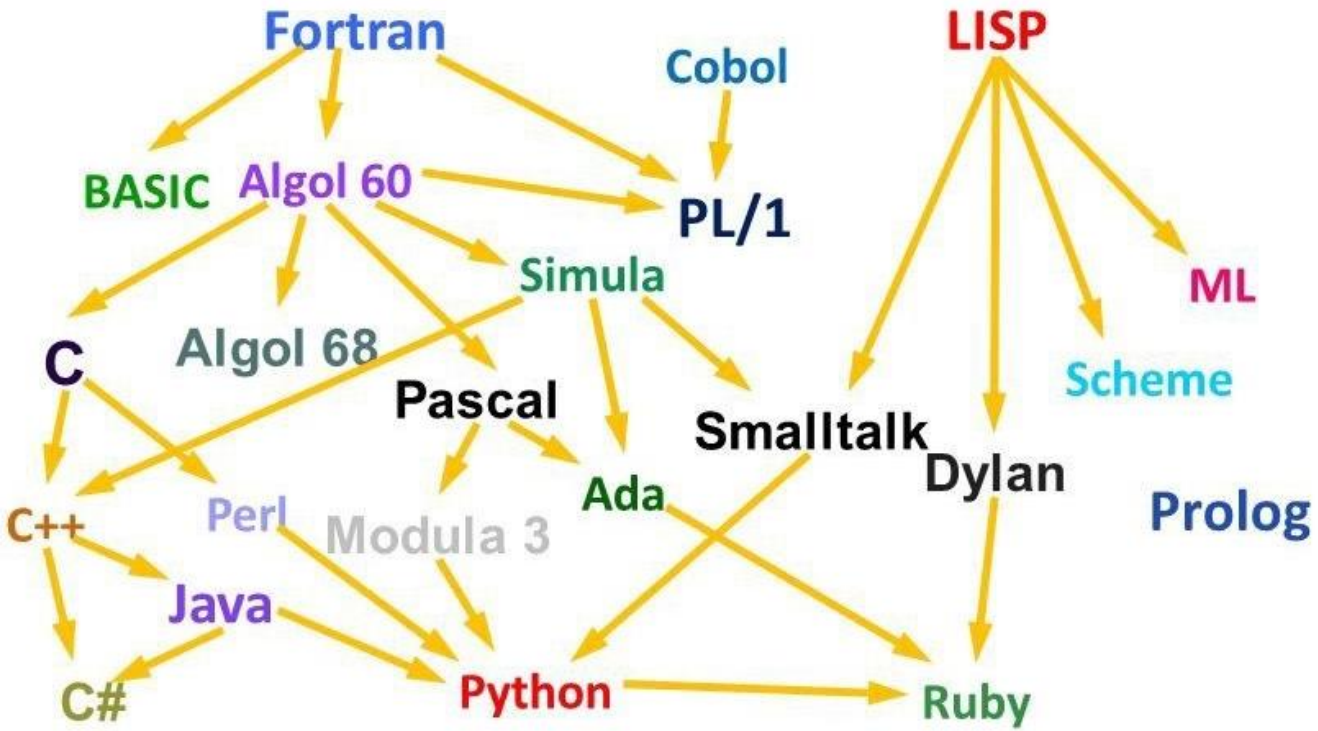
Magdalena Godlewska

na podstawie slajdów prof. J. Daciuka

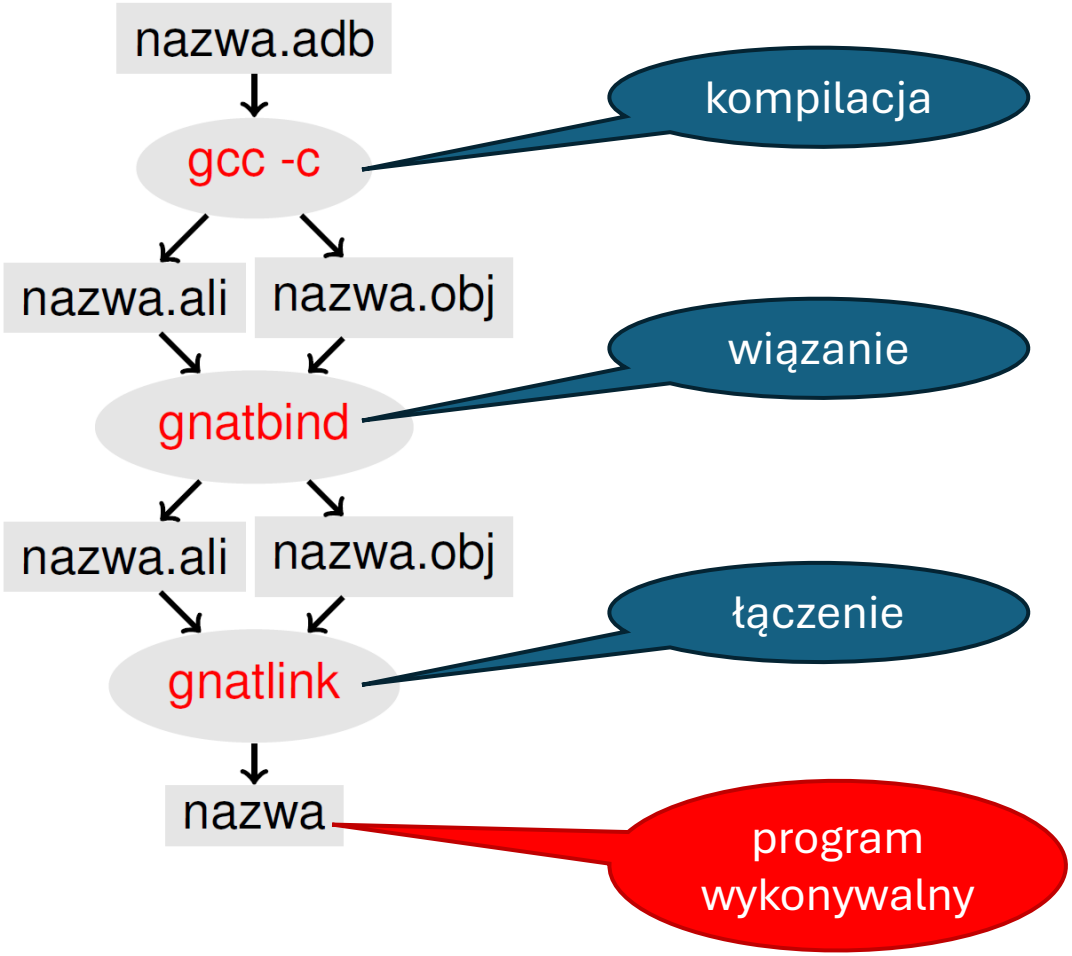
Krótko o historii

- Współbieżny, strukturalny, kompilowany, imperatywny, statycznie typowany język programowania opracowany przez Jean Ichbiaha i zespół z CII Honeywell Bull w latach 70. XX wieku.
- Wygrał konkurs zorganizowany przez Departament Obrony USA (DoD), pokonując 19 innych projektów.
- Wcześniej DoD USA dostrzegł problem wykorzystywania dużej liczby różnych języków programowania używanych dla wewnętrznych systemów wbudowanych. Wiele z nich było przestarzałych, związanych ze sprzętem i nie wspierało bezpiecznego programowania modułowego..
- Nazwa języka, nadana przez DoD, pochodzi od imienia Augusty Ady Lovelace (maszyna analityczna – pierwsza programistka).
- W 1987 roku DoD wydał zarządzenie wymagające użycia Ady w każdym projekcie programistycznym, gdzie nowy kod stanowił więcej niż 30% ogólnego wyniku. Reguła została zniesiona 10 lat później.
- Z uwagi na bezpieczeństwo, Ada jest używana w aplikacjach wojskowych oraz projektach komercyjnych, gdzie błąd programistyczny może mieć kosztowne konsekwencje. Np. kontrola lotów, awionika samolotów, satelity i inne systemy kosmiczne, automatyczne systemy transportowe czy bankowość (Fly-by-wire w samolocie Boeing 777 zostało w całości napisane w Adzie).

Krótko o historii



Kompilacja dla piszących „z palca”



lub

```
~$ gnatmake nazwa.adb
gcc-4.4 -c nazwa.adb
gnatbind -x nazwa.ali
gnatlink nazwa.ali
```

Kompilacja dla piszących „z palca” – program modułowy

producent.ads

```
package Producent is
  procedure Produkuj(...);
  procedure Wyslij(...);
end Producent;
```

konsument.ads

```
package Konsument is
  procedure Przyjmij(...);
  Procedure Konsumuj(...);
end Konsument;
```

producent.adb

```
package body Producent is
...
end Producent;
```

konsument.adb

```
package body Konsument is
...
end Konsument;
```

gmain.adb

```
with...; use...;
procedure Gmain is
...
end Gmain;
```

kompilacja

```
gcc -c gmain.adb
gcc -c producent.adb
gcc -c konsument.adb
```

wiązanie i łączenie

```
gnatbind gmain
gnatlink gmain
```

lub

```
gnatmake gmain
```

Dla preferujących IDE

GNAT Studio | AdaCore

Releases · AdaCore/gnatstudio (github.com)

Dostęp 30.09.2024

Program startowy: simulation.adb
oraz instrukcja uruchomienia go w GPS: GNAT2021_instrukcja.pdf
znajdują się na e-nauczaniu

Typy danych

Silne typowanie, typy na podstawie innych typów

- wyliczeniowy **type X is (A, B,. . .)**
- logiczny **BOOLEAN: FALSE i TRUE**
- całkowite **INTEGER, SHORT_INTEGER i LONG_INTEGER**
- znakowe **CHARACTER**
- rzeczywiste z dokładnością
  - względną **type X is digits n range f1..f2**
  - bezwzględną **type X is delta f range f1..f2**
- podtypy (typy okrojone) **subtype X is Y range**. . .
- rekordy **type X is record. . . end record**
- tablice **type X is array (l..r) of**
- napisy **STRING** (jak tablica znaków)
- wskaźnikowe **type X is access Y**

Operatory

- potęgowanie **
- wartość bezwzględna **abs**
- mnożenie i dzielenie ***, /, rem, mod**
- jednoargumentowe **+, -, not**
- addytywne **+, -**
- sklejanie **&**
- relacyjne **/=, =, >, >=, <, <=**
- przynależności **in**
- logiczne **and, or, xor**
- logiczne warunkowe **and then, or else**

Instrukcje

- przypisania **X := wyrażenie**
- warunkowe **if war1 then. . . elsif war2 then. . . else. . . end if**
- wyboru **case X is when YjZ =>. . . when others =>. . . end case**
- pętli

  loop. . . end loop

  while warunek loop. . . end loop

  for X in zakres loop. . . end loop
- wywołania procedury lub funkcji: parametry identyfikowane przez pozycję lub nazwę

  f(0.5, 0.25)

  f(Y => 0.25, X => 0.5)

Podprogramy

```
procedure nazwa(parametry) is
    deklaracje
begin
    . . .
end nazwa;
```

```
function nazwa(parametry) return typ is
    deklaracje
begin
    . . .
    return wyrażenie;
end nazwa;
```

Parametry:

function f(X, Y : FLOAT; N : INTEGER := 0) **return** FLOAT **is**. . .

rodzaje:

wejściowe (też domyślnie) **in**
wyjściowe **out** (tylko w procedurze)
in out
przez wskaźnik **access**

jeśli nie ma parametrów, opuszczamy nawiasy

Funkcje (a także operatory) można przeciążać.

Zadania

.. czyli fragmenty kodu, które wykonywane są asynchronicznie (tj. niezależnie od siebie), ale od czasu do czasu komunikują się wymieniając pomiędzy sobą dane.

Program zaprojektowany jako wielozadaniowy:

- jest łatwiejszy w konserwacji (małe porcje kodu)
- umożliwia tworzenie systemów ze stopniową degradacją, czyli jeżeli w pewnym zadaniu pojawi się błąd to system jako taki nie przestanie działać, a co najwyżej straci nieco na swojej funkcjonalności
- po błędzie zadania, może ono zostać wznowione, przez co system odzyska swoją funkcjonalność
- na odpowiednim sprzęcie zadania mogą być wykonywane równolegle (współbieżnie), przez co program będzie wykonywany szybciej

Zadania

deklaracja

```
task type Aabb is
    entry APrint;
    entry BPrint;
end Aabb;
```

definicja

```
task body Aabb is
  begin
      accept APrint do
        Put_Line(" AAA ");
      end APrint;

      accept BPrint do
        Put_Line(" BBB ");
      end BPrint;
  end Aabb;
```

Zadania - przykład

```
task type Aaa is
    entry Start;
end Aaa;

task type Bbb is
    entry Start;
end Bbb;

task type Aabb is
    entry APrint;
    entry BPrint;
end Aabb;

A: Aaa;
B: Bbb;
AB: Aabb;
```

```
task body Aaa is
   subtype Aaa_Time_Range
      is Integer range 3 .. 6;
   package Random_Aaa
      is new
Ada.Numerics.Discrete_Random(Aaa_Time_Range);
   G: Random_Aaa.Generator;
   Random_Time: Duration;

 begin
   accept Start do
     Random_Aaa.Reset(G);
   end Start;
   Put_Line("Aaa start");
   loop
      Random_Time :=
Duration(Random_Aaa.Random(G));
     delay Random_Time;

     AB.APrint;

   end loop;
 end Aaa;
```

```
task body Aabb is
  begin
    Put_Line("Aabb started");
    loop

      accept APrint do
        Put_Line(" AAA ");
      end APrint;

      accept BPrint do
        Put_Line(" BBB „);
      end BPrint;

    end loop;
  end Aabb;


begin
  A.Start;
  B.Start;
```

```
Aabb started
Aaa start
Bbb start
 AAA
 BBB
 AAA
 BBB
 AAA
 BBB
 AAA
 BBB
 AAA
 BBB
 AAA
 BBB
 AAA
 BBB
 AAA
 BBB
 AAA
 BBB
 AAA
 BBB
 AAA
```

Spotkania

W Adzie istnieje możliwość bezpośredniej komunikacji zadań za pomocą mechanizmu nazywanego spotkaniem (fr. rendez-vous).

Mechanizm ten polega na tym, że zadanie udostępniające usługę i zadanie chcące z tej usługi skorzystać czekają w pewnym punkcie programu na siebie i następnie wykonywany jest wspólny fragment kodu dla obu tych zadań.

Po wykonaniu tego fragmentu kodu zadania biorące udział w spotkaniu kontynuują swoje działanie asynchronicznie.

Spotkania selektywne oczekiwane

```
task body Aabb is
  begin
    Put_Line("Aabb
started");
    loop
     select
       accept APrint do
         Put_Line(" AAA ");
       end APrint;
     or
       accept BPrint do
         Put_Line(" BBB „);
       end BPrint;
     end select
    end loop;
  end Aabb;
```

```
Aabb started
Aaa start
Bbb start
 AAA
 BBB
 BBB
 AAA
 BBB
 AAA
 AAA
 BBB
 AAA
 BBB
 AAA
 BBB
 BBB
 AAA
 BBB
 AAA
 BBB
 AAA
 BBB
```

Spotkania selektywne z budzikiem/przeterminowaniem

```
task body Aabb is
  begin
    Put_Line("Aabb
started");
    loop
     select
      accept APrint do
        Put_Line(" AAA ");
      end APrint;
     or
      delay 5.0;
      accept BPrint do
        Put_Line(" BBB „);
      end BPrint;
     end select
    end loop;
  end Aabb;
```

```
Aabb started
Aaa start
Bbb start
 BBB
 AAA
 AAA
 BBB
 AAA
 AAA
 AAA
 AAA
 BBB
 AAA
 AAA
 BBB
 AAA
 AAA
 AAA
 AAA
 BBB
 AAA
 AAA
 BBB
 AAA
 BBB
```

Spotkania selektywne z warunkowym wywołaniem

```
task body Aabb is
  begin
    Put_Line("Aabb
started");
    loop
     select
       accept APrint do
         Put_Line(" AAA ");
       end APrint;
     else
         delay 1.0;
         Put_Line (" NUDZE
SIE ");
    end select
    end loop;
  end Aabb;
```

```
Aabb started
Aaa start
Bbb start
 NUDZE SIE
 NUDZE SIE
 NUDZE SIE
 NUDZE SIE
 AAA
 NUDZE SIE
 NUDZE SIE
 NUDZE SIE
 NUDZE SIE
 AAA
 NUDZE SIE
 NUDZE SIE
 NUDZE SIE
 AAA
 NUDZE SIE
 NUDZE SIE
 NUDZE SIE
 NUDZE SIE
 NUDZE SIE
 AAA
 NUDZE SIE
```

Spotkania selektywne z asynchroniczną zmianą wątku sterowania

```
task body Aabb is
  begin
    Put_Line("Aabb
started");
    loop
     select
      accept APrint do
        Put_Line(" AAA ");
      end APrint;
     or
      accept BPrint do
        Put_Line(" BBB „);
      end BPrint;
     end select
    end loop;
  end Aabb;
```

```
task body Aaa is
....
    a : String (1 .. 3);
    b : Integer;
  begin
....
    loop
      Random_Time := Duration
(Random_Aaa.Random (G));
      select
        delay Random_Time;
        AB.APrint;
      then abort
        Put_Line ("teraz nic nie robie, nacisnij
przycisk");
        Get_Line (a, b);
      end select;
    end loop;
  end Aaa;
```

```
Aabb started
Aaa start
teraz nic nie robie, nacisnij przycisk
Bbb start
 BBB
 BBB
 BBB
 BBB

 AAA
teraz nic nie robie, nacisnij przycisk

teraz nic nie robie, nacisnij przycisk

teraz nic nie robie, nacisnij przycisk

teraz nic nie robie, nacisnij przycisk

teraz nic nie robie, nacisnij przycisk
 BBB

 AAA
teraz nic nie robie, nacisnij przycisk
 BBB
 BBB
```

Dozory

Instrukcja **accept** może mieć rozszerzoną składnię używaną w ramach instrukcji **select**, służącą do okresowego blokowania wejścia – nazywa się ją dozorem

when Warunek **=> accept** Nazwa(parametry) **do**

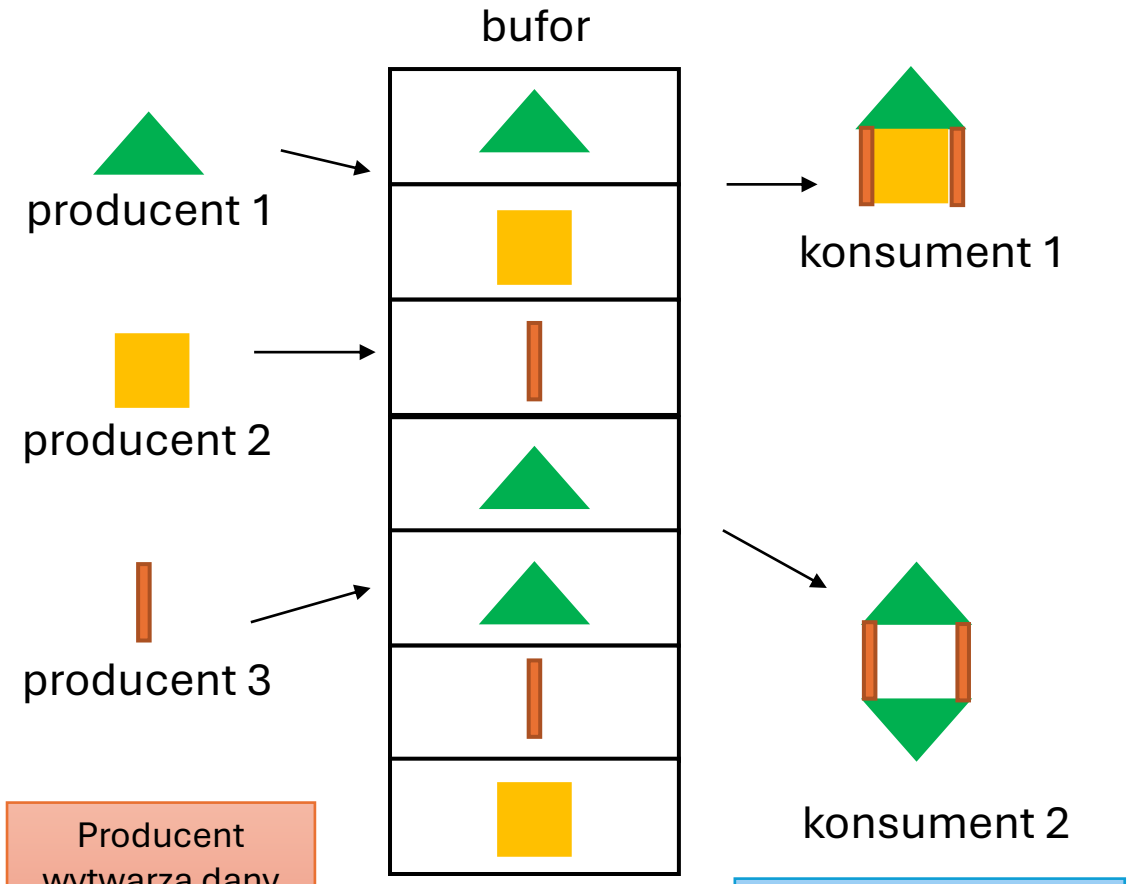
```
task body Aabb is
   subtype Aabb_Range is Integer range 1 .. 6;
   package Random_aabb is new Ada.Numerics.Discrete_Random (Aabb_Range);
   G : Random_aabb.Generator;
   dice: Integer;
 begin
   Put_Line (ESC & "[91m" & "Aabb started" & ESC & "[0m");
   Random_aabb.Reset (G);
   loop
     dice := Random_aabb.Random(G);
     Put_Line("Dice: "  & Integer'Image(dice));

     select
       when dice=6 => accept APrint do
         Put_Line (ESC & "[95m" & " AAA " & ESC & "[0m");
       end APrint;

     or
       accept BPrint do ....
```

```
Aabb started
Dice:  2
Aaa start
Bbb start
 BBB
Dice:  4
 BBB
Dice:  5
 BBB
Dice:  2
 BBB
Dice:  1
 BBB
Dice:  2
 BBB
Dice:  1
 BBB
Dice:  5
 BBB
Dice:  6
 AAA
Dice:  2
 BBB
```

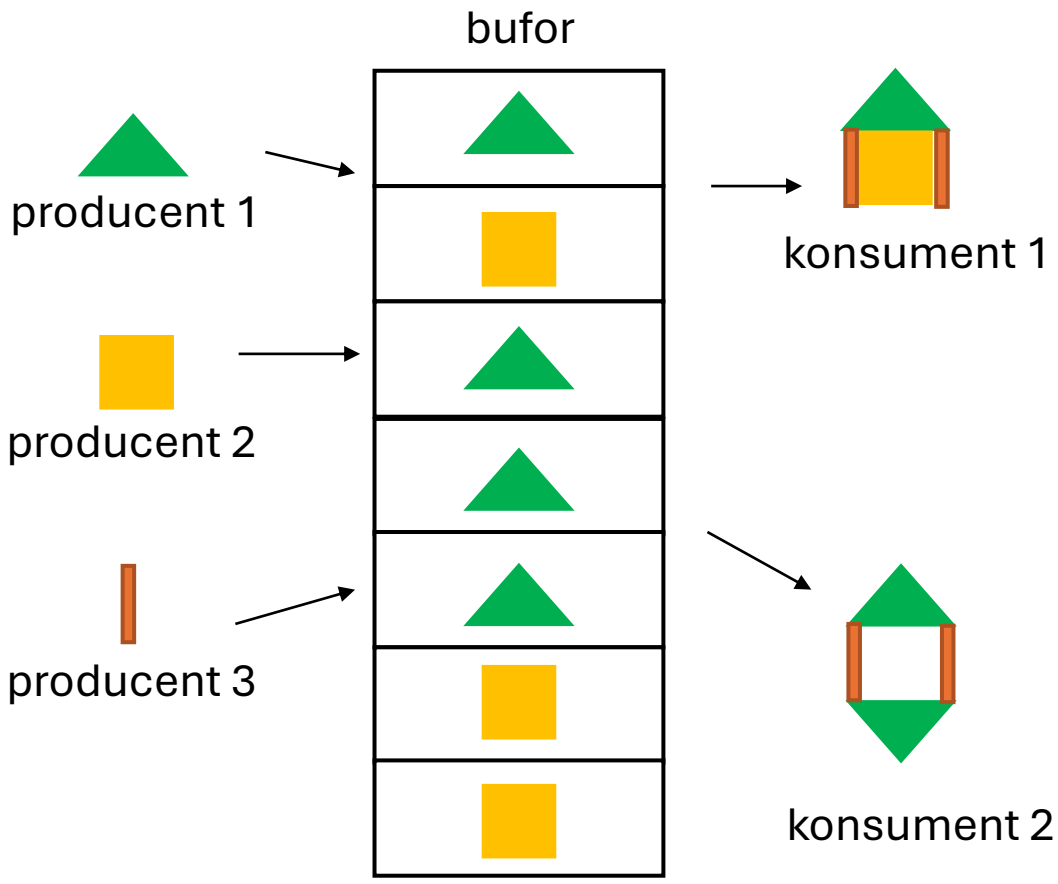
O co chodzi w projekcie?



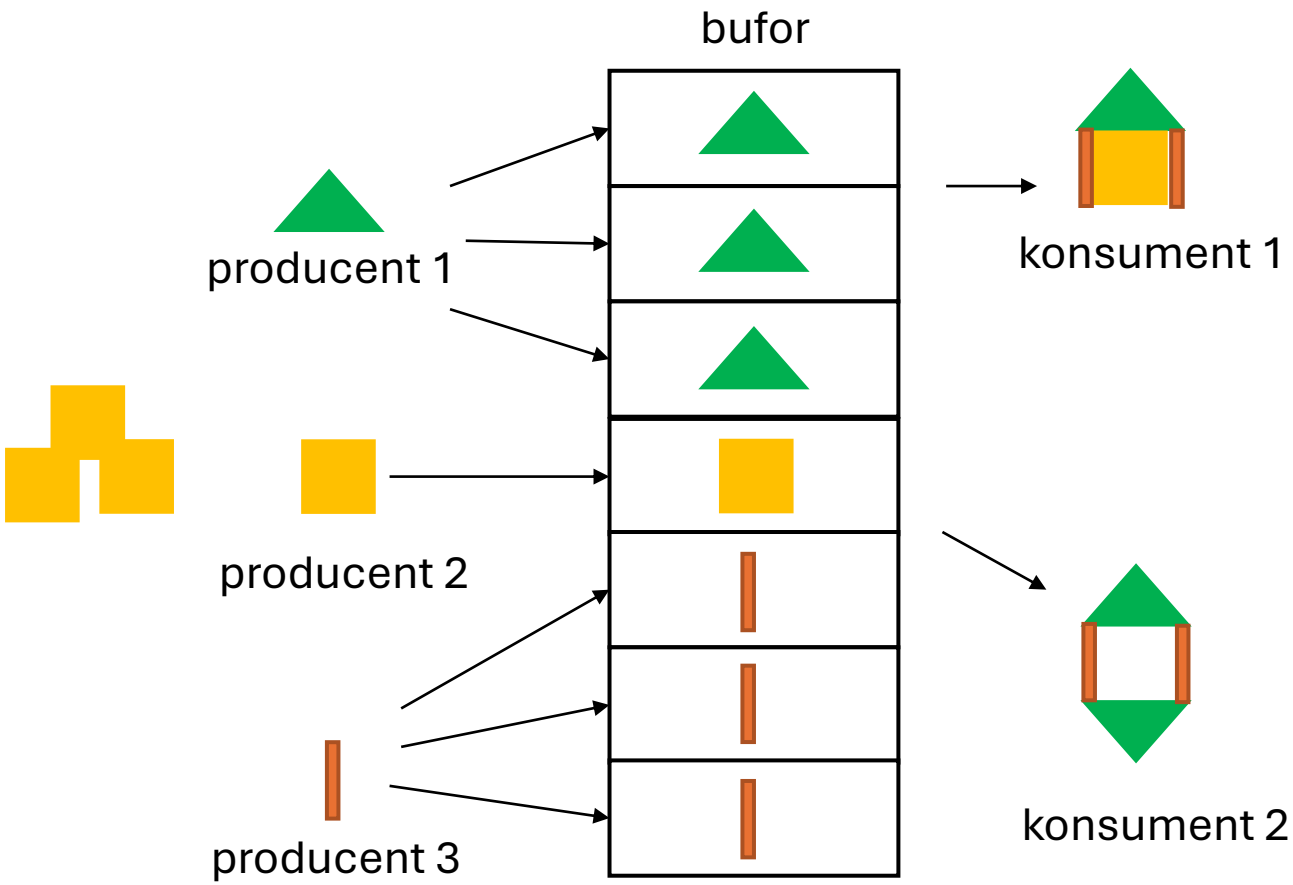
Główne punkty do rozwiązania:

1. Wybór tematyki
2. Zastosowanie spotkań
3. Rozwiązanie problemów brzegowych, C: Consumer2 takes assembly Assembly1 number 0
4. Rozwiązanie typowych problemów współbieżności
5. Zrozumienie programu

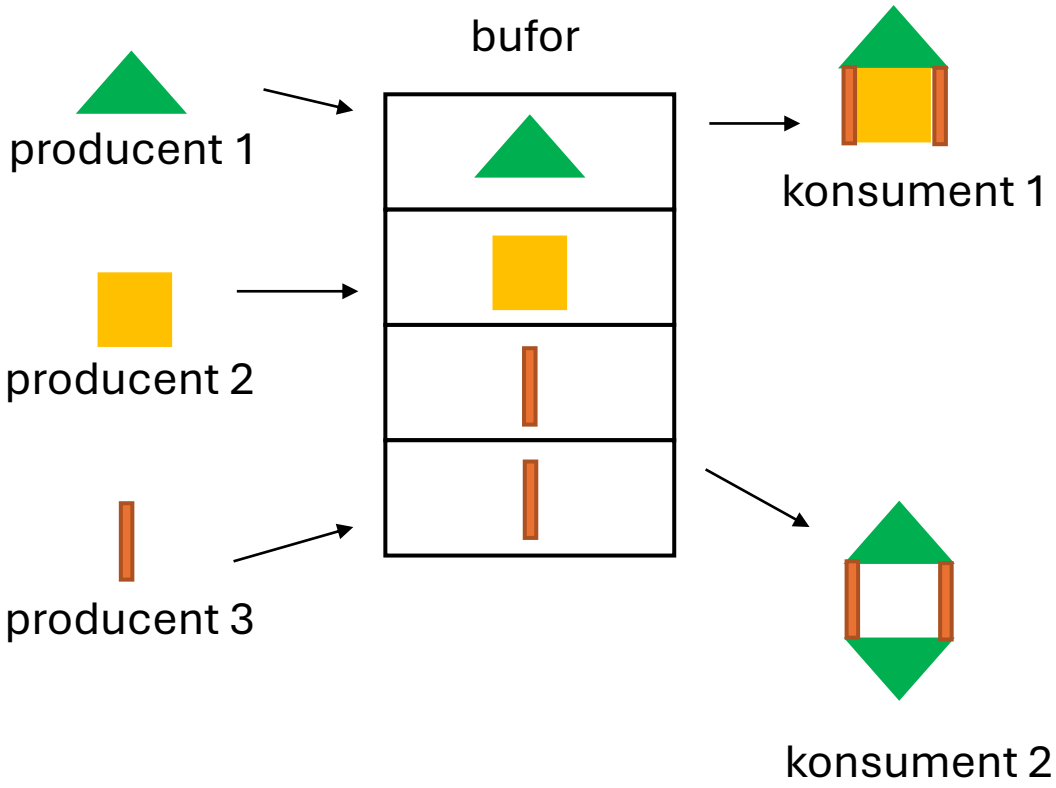
Problemy współbieżności – zakleszczenie (deadlock)



Problemy współbieżności - zagłodzenie



Problemy współbieżności – wąskie gardło



Treść zadania

1. Należy dobrać się w 2-3 osobowe zespoły i zapisać się do jednej z grup na e-nauczaniu
2. Wówczas udostępni się dla danej grupki treść zadania.
3. Jedna z osób w grupie wygrywa na e-nauczanie rozwiązanie zadania do **20.10.2024** i w komentarzu wpisuje osoby tworzące grupkę.
4. Na e-nauczaniu będzie zamieszczony harmonogram oddawania projektów. W wyznaczonym terminie proszę całą grupkę o przyjście i demonstrację rozwiązania (ok 10 min).