# Ada 95 – pierwsze spotkanie

**Wojciech Complak,**
***Instytut Informatyki, Politechnika Poznańska***
***e-mail :*** ***Wojciech.Complak@cs.put.poznan.pl***
***www :*** ***http://www.cs.put.poznan.pl/wcomplak***

**2006-02-27, v. 0.24**

## Plan wykładu

- Ada Augusta Byron/King hrabina Lovelace
- Geneza i historia języka Ada
- Przykładowe zastosowania
- Zalety i wady Ady
- Struktura elementarnego programu
- Kompilacja i uruchamianie programu
- Instrukcje i podstawowe operacje wejścia/wyjścia

## Ada Augusta Byron (King), hrabina Lovelace (#1/2)

- Ada Augusta Byron, córka Lorda Byrona, urodzona 10-12-1815, zmarła 27-11-1852 w wieku 36 lat na raka
- utalentowana matematycznie, kształciła się m. in. u Augustusa DeMorgana (twórca praw algebry Booleowskiej)
- Ada współpracowała z Charlesem Babbagem pomagając w dokumentowaniu jego projektów, tłumacząc jego prace i projektując programy wykorzystywane w jego maszynach

## Ada Augusta Byron (King), hrabina Lovelace (#2/2)

- w latach 1842-1843 (w dziewięć miesięcy), Ada przetłumaczyła dla Babbage'a rozprawę włoskiego matematyka o nazwisku Louigi Menebrea na temat najnowszej konstrukcji Babbage'a – maszyny analitycznej („The Sketch of the Analytical Engine")
- do artykułu dołączyła zbiór własnych notatek (dwukrotnie dłuższy niż oryginał), które opisywały m. in. szczegółowo metodę obliczania liczb Bernoulliego za pomocą maszyny, algorytm ten jest uważany za pierwszy program komputerowy
- na jej cześć US DoD nadał nazwę nowemu językowi programowania

## Geneza i historia języka Ada (Ada 83, Ada 95, Ada 2005) (#1/3)

- w latach 60-tych i 70-tych US DoD używał ponad 2000 języków do oprogramowania klasy „mission critical" (w większości zaprojektowanych do jednego, konkretnego zadania)
- w 1975 powołano grupę U.S. Department of Defense High-Order Language Working Group (HOLWG) do rozwiązania sytuacji określanej jako „software crisis"
- grupa HOLWG dążąc do stworzenia uniwersalnego języka do programowania systemowego, sztucznej inteligencji, a szczególnie systemów czasu rzeczywistego i systemów wbudowanych rozpisała konkurs na język programowania

## Geneza i historia języka Ada (Ada 83, Ada 95, Ada 2005) (#2/3)

- 1977 – wstępny wybór czterech języków
- 1978 – liczbę kandydatów ograniczono do dwóch (oba projekty to języki Pascalo-podobne)
- 1979 – ostatecznym zwycięzcą konkursu zostaje zespół CII Honeywell-Bull z Francji (zespół zielony – zespoły uczestniczące w konkursie oznaczano kolorami)
- 1979 – pierwszy standard roboczy przygotowany przez DoD, język zmienia nazwę z DoD-1 na Ada (autorem nazwy jest Jack Cooper z Navy Material Command)

## Geneza i historia języka Ada (Ada 83, Ada 95, Ada 2005) (#3/3)

- 1983 – oficjalny standard języka (DoD i ANSI wydają „Reference Material for the Ada Programming Language")
- 1990 – istnieje ponad 200 zgodnych ze standardem kompilatorów
- 1995 – nowy standard Ady – Ada 95, język obiektowo zorientowany, interfejsy do C, Fortranu i Cobola, „stara" Ada od tego momentu jest nazywana „Ada 83"
- 2006 – Ada 2005, profil Ravenscar, Java-podobny interfejsy, wsparcie dla 32-bitowych znaków, nowe biblioteki standardowe

## Przykładowe zastosowania

- awionika Boeinga 777 (99% oprogramowania w Adzie)
- system monitorowania w samolocie Airbus A340 (Flight Warning System)
- systemy pilotażu samolotów bezzałogowych (Szwajcaria - Unmanned Air Vehicles i Remotely Piloted Vehicles)
- koleje i metra (Londyn, Nowy Jork, Paryż, Kair, Kalkuta, koleje francuskie i TGV,
- w systemach informacyjnych i telekomunikacji (europejski GPS)
- w sterowaniu systemami przemysłowymi (Pratt-Whitney – silniki do samolotów wojskowych, Weirton Steel Corporation - metalurgia)

## Zalety Ady (#1/2)

- uniwersalny język programowania (od systemów wbudowanych do systemów rozproszonych) przeznaczony głównie do programowanie w dużej skali
- język o jakości produkcyjnej (ścisła kontrola typów, wykrywanie błędów na etapie kompilacji, wykrywanie i obsługa błędów w trakcie wykonania)
- programowanie systemowe (dostęp do sprzętu, wywołań systemowych, obsługa przerwań, sterowanie reprezentacją i alokacją danych, dostęp do zmiennych współdzielonych)
- wbudowane mechanizmy programowania współbieżnego i szeregowania zadań

## Zalety Ady (#2/2)

- język wspierający programowanie zorientowane obiektowo
- silna modularyzacja i hermetyzacja
- wsparcie dla programowania hybrydowego i wykorzystania kodu „legacy" (C, Fortran, Cobol, asembler)
- międzynarodowy, uznany standard gwarantujący przenośność oprogramowania (praktycznie wszystkie platformy sprzętowe i programowe)
- radykalna poprawa niezawodności oprogramowania
- redukcja kosztów wytwarzania i utrzymania oprogramowania
- wysoka wydajność kodu

## Wady Ady

- stosunkowo skomplikowany język – dużo słów kluczowych i redundantna składnia (w porównaniu np. do C)
- długi czas nauki (zrozumienia filozofii) języka
- duży koszt w przypadku małych projektów
- wydłużony czas implementacji

## Elementarny program w Adzie (#1/2)

```
procedure test is
begin
  null;
end test;
```

- programy tworzymy za pomocą dowolnego edytora tekstowego
- wielkość liter jest nieistotna
- na końcu jednostki (procedury) opcjonalnie powtarzamy jej nazwę
- *null* to instrukcja pusta
- komentarze rozpoczynają się od „--" i ciągną się do końca linii

## Elementarny program w Adzie (#2/2)

```
procedure test is
begin
  null;
end test;
```

- BARDZO WAŻNA jest ścisła zależność między nazwami jednostek kompilacji, zawartością plików a nazwami plików:
  - rozszerzenie „ads” = specyfikacja, interfejs („spc”)
  - rozszerzenie „adb” = ciało, implementacja („bdy”)
  - rozszerzenie „ada” = kompletny program
  - nazwa główna odpowiada nazwie jednostki kompilacji
  - niektóre kompilatory nie egzekwują standardu (OA)

## Kompilacja programu w Adzie (krok po kroku)

- program zapisujemy w pliku test.adb

```
procedure test is
begin
  null;
end test;
```

- kompilujemy

```
gcc -c test.adb
```

- tworzymy plik wykonywalny

```
gnatbind test
gnatlink test
```

## Kompilacja programu w Adzie (gnatmake)

- program zapisujemy w pliku test.adb

```
procedure test is
begin
  null;
end test;
```

- tworzymy plik wykonywalny

```
gnatmake test.adb
```

## Odpluskwianie programu w Adzie (gdb)

- budujemy z informacjami do odpluskwiania

```
gnatmake -g test.adb
```

- uruchamiamy debugger

```
gdb test
```

```
c:\test>gdb test
GNU gdb 5.0.gnat.3.15p
Copyright 2000 Free Software Foundation, Inc.
Ada Core Technologies version of GDB for GNAT Professional
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
See your support agreement for details of warranty and support.
If you do not have a current support agreement, then there is absolutely
no warranty for this version of GDB. Type show warranty for details.
This GDB was configured as "i686-pc-cygwin"...
(gdb)
```

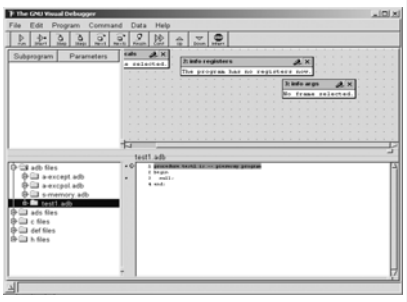
## Odpluskwianie programu w Adzie (gvd)

- budujemy z informacjami do odpluskwiania

```
gnatmake -g test.adb
```

- uruchamiamy debugger

```
gvd test
```



## Deklaracje

- ogólna postać

```
Nazwa_Zmiennej [, Nazwa_Zmiennej]*:Nazwa_Typu;
```

- przykładowe deklaracje zmiennych prostych

```
A    : Integer;
B, C : Character;
M    : Float := 1.34;
Days : Positive range 1 .. 7;
```

- przykładowe deklaracje tablic

```
tab  : array (1 .. 10) of Integer;
mtab : array (1..2,1..2) of Positive;
```

- przykładowe odwołania do tablic

```
tab(2) := mtab(1,1);
```

## Podstawowe operatory

| | |
|---|---|
| **logiczne** | `: and, or, xor` |
| **relacyjne** | `: =, /= (różne), <, <=, >, >=` |
| **przypisanie** | `: :=` |
| **arytmetyczne** | `: +, -, *, /, mod, rem, **, abs, not` |

## Pętla *for*

• ogólna postać

```
[Identyfikator:]
for Identyfikator in [reverse] Zakres loop
  instrukcja/instrukcje
end loop [Identyfikator];
```

• uwaga – iteratora nie trzeba deklarować (jest wtedy zmienną lokalną pętli deklarowaną niejawnie)

• przykład :

```
petla1:
for z1 in 1 .. 20 loop
  Put(z1);
  Put_Line("");
end loop petla1;
```

```
for z1 in 1 .. 20 loop
  Put(z1);
  Put_Line("");
end loop;
```

## Pętla *while*

• ogólna postać

```
[Identyfikator:]
while Warunek loop
  instrukcja/instrukcje
end loop [Identyfikator];
```

• do przerywania pętli można wykorzystać instrukcję *exit [identyfikator]*

• *instrukcja exit* wraz z identyfikatorami pętli pozwala wychodzić z zagnieżdżonych pętli

• przykład niezagnieżdżonej pętli:

```
petla1:
while x <= 10 loop
  x := x + 1;
end loop petla1;
```

```
while x <= 10 loop
  x := x + 1;
end loop;
```

## Instrukcja warunkowa *if* (#1/2)

• ogólna postać

```
if Warunek then
  instrukcja/instrukcje
end if;
```

```
if Warunek then
  instrukcja/instrukcje
else
  instrukcja/instrukcje
end if;
```

## Instrukcja warunkowa *if* (#2/2)

• przykład :

```
if a > b then
  c := a;
else
  c := b;
end if;
```

## Blok instrukcji

• ogólna postać

```
[Identyfikator:]
  declare
    [deklaracje lokalne]
  begin
    instrukcja/instrukcje
  end [Identyfikator];
```

• przykład :

```
swap:
  declare
    t : integer;
  begin
    t:=a; a:=b; b:=t;
  end swap;
```

```
declare
    t : integer;
begin
    t:=a; a:=b; b:=t;
end;
```

## Procedury (#1/2)

• ogólna postać

```
procedure Nazwa[(parametry_formalne)] is
  [deklaracje_lokalne]
begin
  instrukcja/instrukcje
end Nazwa;
```

• parametry formalne

```
(par [,par]*:tryb typ;[par [,par]*:tryb typ;]*)
```

• tryby przekazywania parametrów

```
in     – parametr wejściowy (domyślny)
in out – parametr wejściowy/wyjściowy
out    - parametr wyjściowy
```

## Procedury (#2/2)

• przykładowa procedura

```
procedure SortAscending(a,b:in out Integer) is
  begin
    if a > b then
      swap: declare
        temp : integer;
      begin
        temp := a; a := b; b := temp;
      end swap;
    end if;
  end SortAscending;
```

• wywołanie

```
SortAscending(x,y);
```

## Podstawowe operacje wejścia wyjścia

• wyjście tekstowe

```
with Ada.Text_IO; use Ada.Text_IO;
procedure test1 is
begin
  Put("Ada "); Put_Line("is the best");
end test1;
```

• wejście/wyjście dla liczb całkowitych

```
with Ada.Integer_Text_IO;
use Ada.Integer_Text_IO;
procedure test2 is
  x : integer;
begin
  Get(x); Put(x);
end test2;
```

## Program obliczający n-tą liczbę Fibonacciego

```
with Ada.Text_IO, Ada.Integer_Text_IO;
use  Ada.Text_IO, Ada.Integer_Text_IO;
procedure Fibo is    -- drukuje n-tą liczbę Fibonacciego
   N : Integer;
begin
   Put("Podaj n = ");   Get(N);
   if N < 0 then Put("bledna wartosc n");
   else if N < 2 then Put(1);
         else declare
                F0, F1, F2 : Integer;
              begin
                F0 := 1;   F1 := 1;
                for I in 2 .. N loop
                  F2 := F0 + F1;  F1 := F0;  F0 := F2;
                end loop;
                Put(F2);
              end;
        end if;
   end if;
end Fibo;
```