Języki programowania: wprowadzenie do Ady

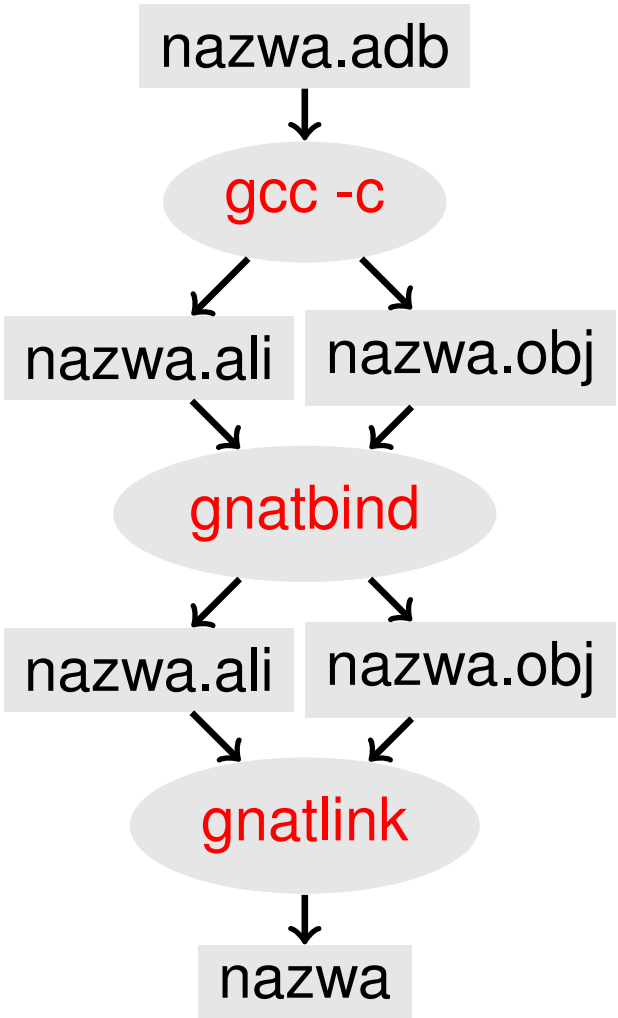
Jan Daciuk

21 września 2019

Materiały dodatkowe

- Ada95 Reference Manual /usr/share/doc/ada-reference-manual/html/aarm95tc1/AA-TOC.html
- Ada Syntax Diagrams http://cui.unige.ch/isi/bnf/Ada95/BNFindex.html
- Ada Programming http://en.wikibooks.org/wiki/Ada_Programming
- GNAT User's Guide `/usr/share/doc/gnat-4.4-doc/gnat_ugn.html`
- Using the GNAT Programming Studio /usr/share/doc/gnat-gps/html/index.html

Kompilacja – prosty program w jednym pliku



plik źródłowy

kompilacja

Ada library information, obj

wiązanie

Ada library information, obj

łączenie

program wykonywalny

Kompilacja – prosty program w jednym pliku

Wydawanie tylu poleceń nie jest konieczne. Jedno polecenie załatwia wykonanie wielu:

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

Po wykonaniu polecenia otrzymamy plik wykonywalny.

Kompilacja – kilka modułów

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 – kilka modułów

Kompilujemy moduły w dowolnej kolejności:

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

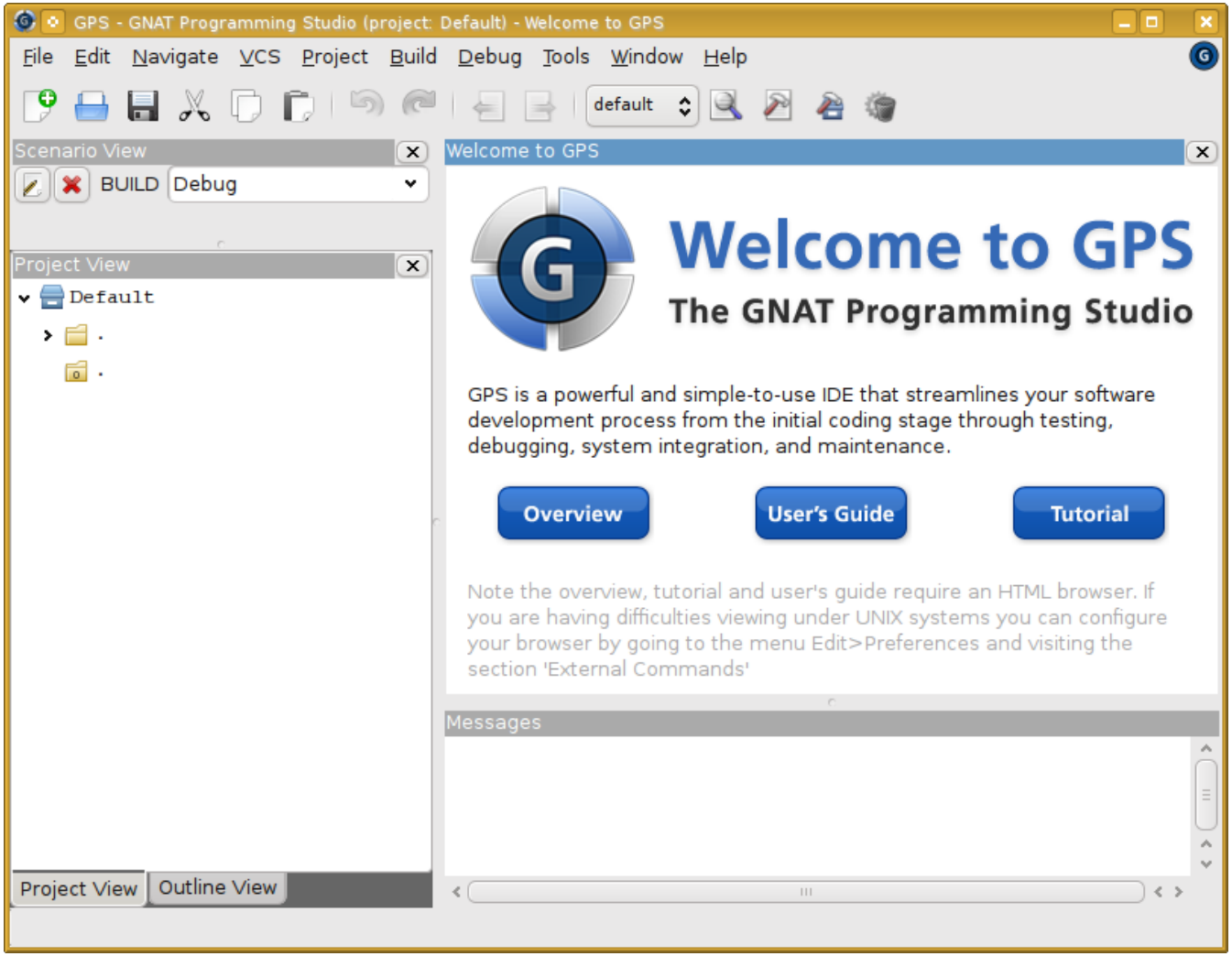
Następnie je wiążemy i łączymy:

```
gnatbind gmain
gnatlink gmain
```

Można też sporo zaoszczędzić pisząc po prostu:

```
gnatmake gmain
```

GNAT Programming Studio



Typy

- wyliczeniowy **type** X **is** (A, B,. . .);
- logiczne 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** Y|Z =>...**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

Przeciążanie

Może istnieć kilka funkcji lub procedur o tej samej nazwie, ale różnych parametrach (jak w C++).

Można także przeciążać operatory, np.
function "*"(X: MATRIX; Y: COL_VEC) **return** COL_VEC **is**
...
end "*";

Wyjątki

```
wyjatek: exception;                          ← deklaracja wyjątku
...
begin
    ...
    raise wyjatek;                           ← zgłoszenie wyjątku
    ...
exception
    when wyjatek =>
        ...                                  ← obsługa wyjątku
end;
```

Wyjątki mogą być też zgłaszane przez standardowe biblioteki.

Pakiety

Interfejs pakietu zawiera stałe, zmienne, typy, funkcje, procedury itp. udostępniane przez pakiet innym bytom w programie. Deklaracje zamyka się w konstrukcji:

package *nazwa* **is**
interfejs pakietu (stałe, procedury itp.)
end *nazwa*;

Tak zadeklarowany interfejs pakietu umieszcza się w pliku o rozszerzeniu `.ads`.

Pakiety

Aby skorzystać z pakietu, trzeba zgłosić chęć skorzystania z niego:

with X, Y;

Wówczas można korzystać ze stałych, zmiennych, typów, procedur, funkcji itp. dostarczanych przez pakiety X i Y poprzedzając identyfikatory tych elementów nazwą pakietu i kropką, np. X.Zmienna.

W przypadku braku konfliktu nazw, można nazw z tych pakietów używać bez nazwy pakietu i kropki, jeżeli dostarczy się dyrektywę:

use X, Y;

Działa to trochę jak **using namespace** w C++.

Pakiety

Ciało (treść) pakietu jest zawarta w konstrukcji:

package body *nazwa* **is**
treść pakietu
end *nazwa*;

Deklaracje i definicje zawarte wewnątrz mają zasięg miejscowy, o ile nie zostaną uwidocznione w interfejsie pakietu.

Zadania

Zadania także są najpierw deklarowane, potem definiowane:

task zadanie;

task body zadanie **is**
    treść zadania
end zadanie;

Deklaracja i definicja mają miejsce w jednostce macierzystej, np. bloku **declare** lub procedurze.

Współbieżność

```
task bufor is
   entry Pisz(C: CHARACTER);
   entry Czytaj(C: out CHARACTER);
end bufor;
```

```
task body bufor is
   CH: CHARACTER;
begin
   loop
      accept Pisz(C: CHARACTER) do
         CH := C;
      end Pisz;
      accept Czytaj(C: out CHARAC-
TER) do
         C := CH;
      end Czytaj;
      exit when CH = ASCII.EOT;
   end loop;
end bufor;
```

Niedeterminizm w zadaniach

Spotkania selektywne (wywołanie dowolnego wejścia):

```
select
  accept ...
    ...
  end ...;
or
  accept ...
    ...
  end ...;
end select
```

Części **or** może być więcej, natomiast zamiast **accept** może wystąpić instrukcja **delay** określająca przeterminowanie po stronie odbiorcy lub instrukcja **terminate** oznaczająca zakończenie zadania (wtedy zadanie może się skończyć w każdej chwili).

Niedeterminizm w zadaniach

Czasomierz z budzikiem (zadanie musi być wywołane w określonym czasie, lub podjęte zostaną środki zaradcze):

select
  wywołanie zadania
or delay *sekund*
  działanie zastępcze
end select;

Gałęzi z instrukcjami **accept** może być więcej.

Niedeterminizm w zadaniach

Spotkania warunkowe (wywołanie zadania lub podjęcie działań zastępczych, jeśli zadanie nie może być wykonane natychmiast):

select
  wywołanie zadania
else
  działanie zastępcze
end select;

Niedeterminizm w zadaniach

Warunkowe zakończenie zadania (jeżeli część główna jest wstrzymana, np. na skutek wywołania wejścia lub instrukcji **delay**, zaczyna wykonywać się część poboczna; zakończenie jednej części kończy drugą):

select
  część główna
then abort
  część poboczna
end select;

Niedeterminizm w zadaniach

Każda gałąź instrukcji **select** może być poprzedzona dozorem, który ma postać:

when *warunek* **=>**

Kiedy warunek ma wartość FALSE, gałąź nie jest wykonywana.