ADA

Декларација процедуре

- Процедуре се декларишу на следећи начин:

procedure <Procedure> (листа формалних параметара) is

декларације

begin

листа наредби

end <Procedure>;

Декларација функције

Функције се декларишу слично као процедуре:

function <Function> (листа формалних параметара)
  return <Type> **is**
  декларације
begin
  листа наредби
end <Function>;

Улазно-излазни параметри

Улазни, излазни и улазно-излазни параметри се декларишу на следећи начин:

<Parameter>: [**in**|**out**|**inout**] <Type>

Декларација пакета

package <Package> is

    јавне декларације

end <Package>;

package body <Package> is

    локалне декларације

    дефиниције јавно декларисаних потпрограма и таскова

end <Package>;

Декларација процеса

```
task <Task> is
   декларације тачака улаза
end;
task body <Task> is
   локалне декларације
begin
   листа наредби
end <Task>;
```

Рандеву

entry <Entry>(листа формалних параметара)

Тачка позива изгледа слично као позив процедуре: именује се процес, тачка улаза и прослеђују стварни параметри:

call <Task>.<Entry>(листа стварних параметара)

Тачка прихватања позива је место у коду серверског процеса где се чека на позив од стране клијентског процеса (преко специфициране тачке улаза):

accept <Entry> do

листа наредби

end;

Селективна наредба

```
select
    [when condition1 =>] selective_wait_alternative
or
    ...
or
    [when conditionн =>] selective_wait_alternative
[else
    sequence_of_statements]
end select;
```

Селективна наредба

selective_wait_alternative ::= accept_alternative |
delay_alternative |
terminate_alternative

accept_alternative ::=
accept_statement[sequence_of_statements]

delay_alternative ::= delay_statement
[sequence_of_statements]

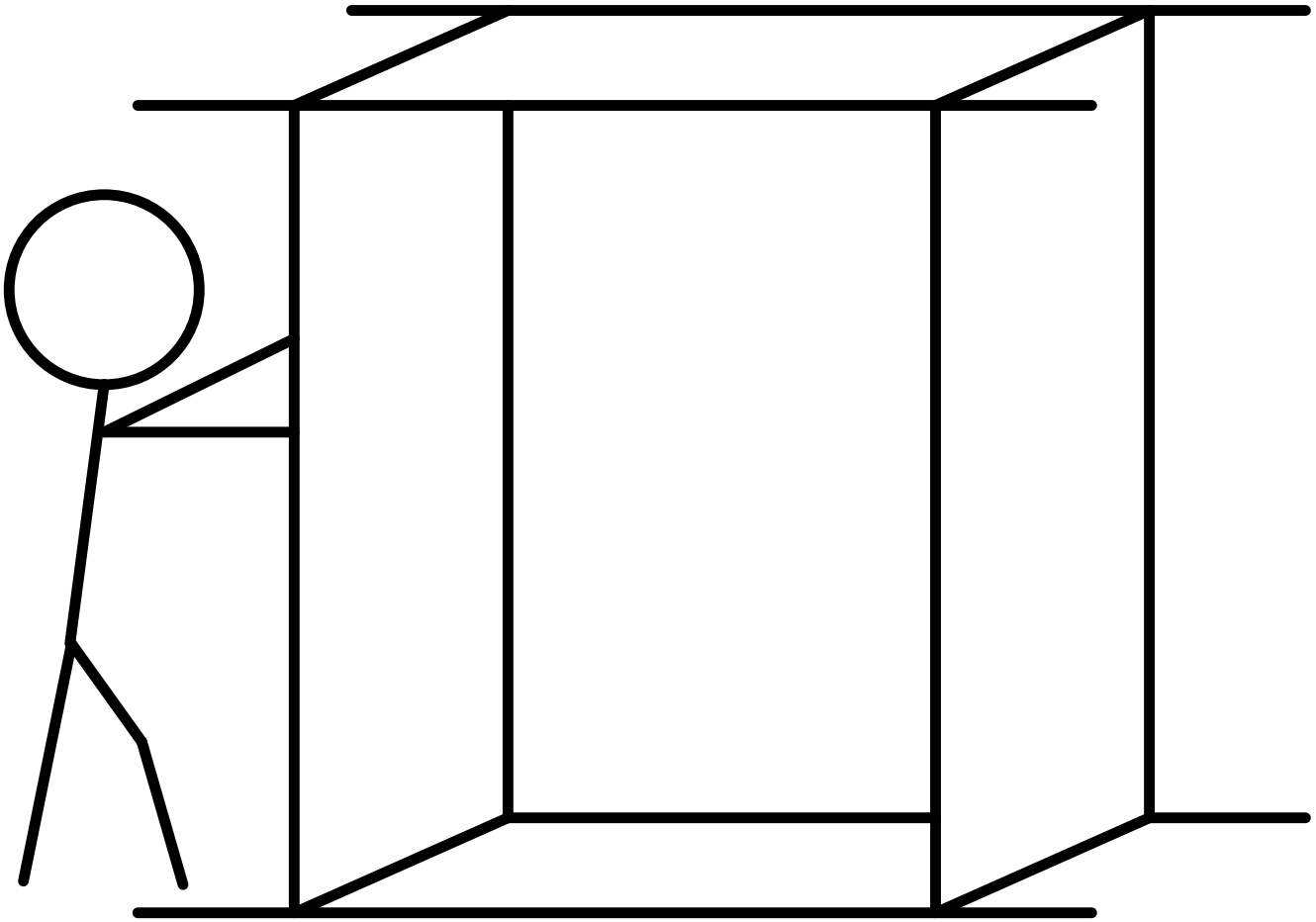
terminate_alternative ::= terminate;

Превођење

```
gcc -c imefajla.adb
gnatbind imefajla
gnatlink imefajla
gnatmake imefajla.adb
```

Задаци

Једноелементни прихватник



Једноелементни прихватник

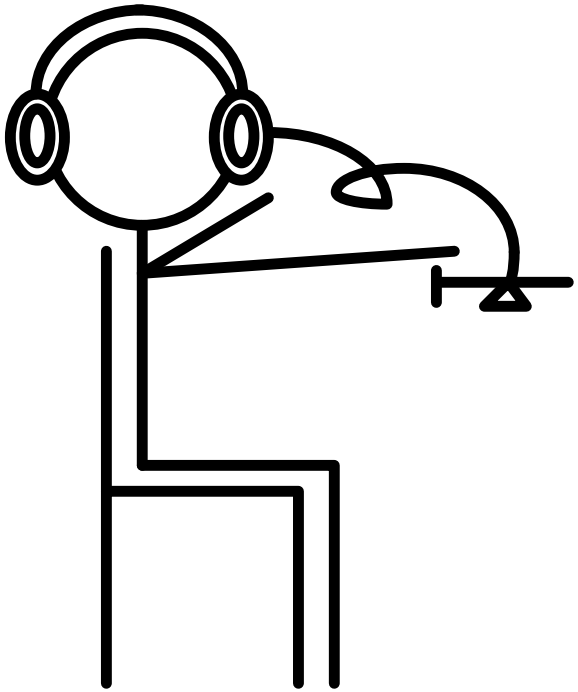
Процедура P у језику Ada садржи локални једноелементни прихватник облика

```
task BOX is
    entry PUT(X: in integer);
    entry GET(X: out integer);
end;
task body BOX is
        V : INTEGER
begin
    loop
        accept PUT(X: in integer) do
            V := X;
        end;
        accept GET(X: out integer) do
            X := V;
        end;
    end loop;
end BOX;
```

Копирање поруке

```
task BOX is
entry PUT(X: in integer);
entry GET(X: out integer);
end;
task body BOX is
    V : INTEGER
begin
loop
    select
        accept PUT(X: in integer) do
            V := X;
        end;
    or
        terminate;
    end select;
    select
        accept GET(X: out integer) do
            X := V;
        end;
    or
        terminate;
    end select;
end loop;
end BOX;
```

Комуникација са поузданим везама



Комуникација са поузданим везама

- Један дистрибуирани оперативни систем обезбеђује за интерпроцесну комуникацију поуздане операције асинхроног слања и временски условљеног пријема облика
- procedure snd(P: process, m: msg)
- function rcv(P: process, var m: msg, t: duration): boolean
- где тип порука msg обухвата целе бројеве и специјалне симболе као што су ack и nack за позитивне односно негативне потврде (по потреби можете увести и друге симболе), временски тип duration обухвата вредности у опсегу 0..maxtime и специјалну вредност ∝ која означава неограничено чекање, а rcv враћа булову вредност која показује да је ли порука примљена у назначеном интервалу времена t.
- Користећи Pascal проширен наведеним операцијама, реализовати следеће бидирекционе трансакције типа rendez-vous, са семантиком као у језику Ada, између процеса S који тражи услугу SERV и процеса R који му је нуди. Именовање не треба решавати на генералан начин, већ сматрати да су ово једини процеси и да знају један за другог.

Комуникација са поузданим везама

```
{ proces S }
A,B: integer:= 1;
…
R.SERVE(A,B);
```

```
{ proces R }
Accept SERVE(X:in integer;
Y:out integer) do
Y := F(X);
end SERVE;
```

(b)

```
{ process S }
A,B: integer := 1;
…

select
    R.SERVE(A,B);
    NORMAL_ACTION;
or
    delay t1;
    TIMEOUT_ACTION;
end select;
```

```
{ proces R }
select
accept SERVE(X:in integer; Y:out
integer) do
        Y := F(X);
    end SERVE;
else
    DEFAULT_ACTION;
end select;
```

Комуникација са поузданим везама

```
var a, b: integer;
    st: boolean
begin
    …
    a := 1;
    snd(R, a);
    st := rcv(R, b, ∝);
    …
end
```

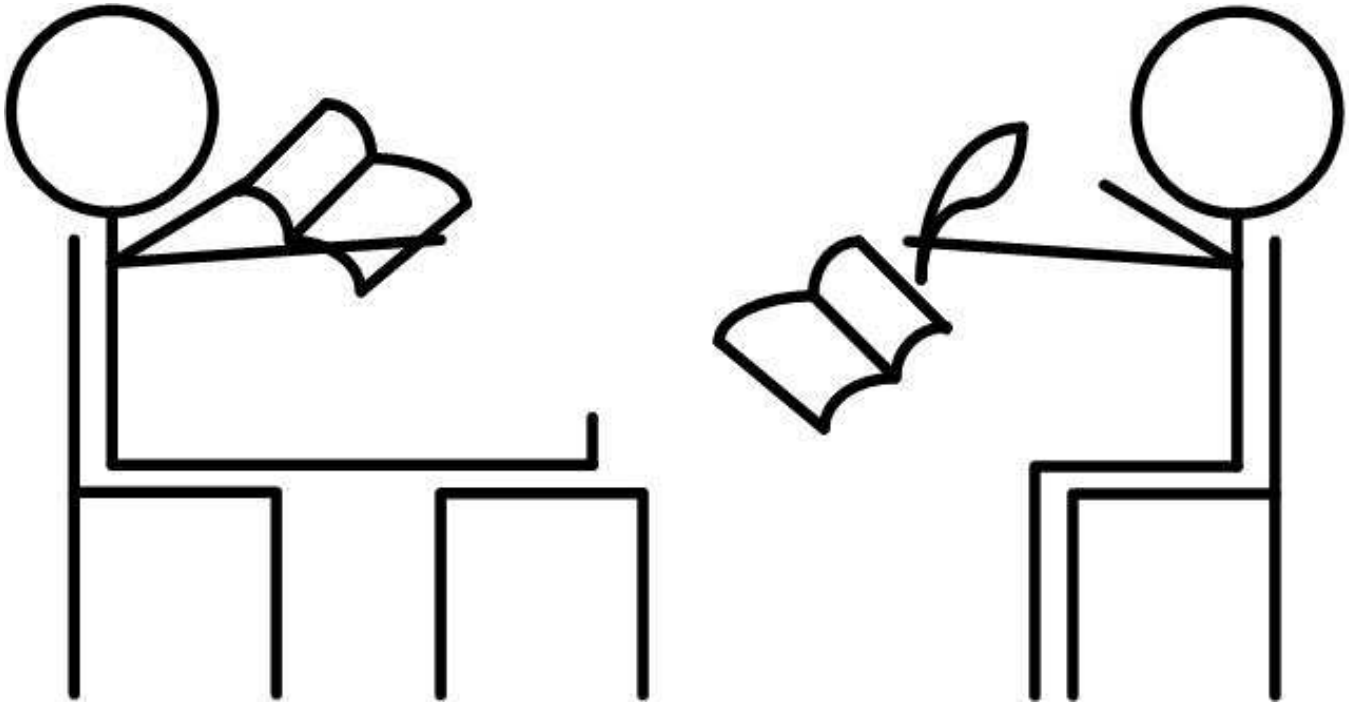
```
var x, y: integer;
    st: boolean;
begin
    …
    st := rcv(S, x, ∝);
    y := f(x);
    snd(S, y);
    …
end
```

Комуникација са поузданим везама

```
var a, b: integer;
    st: boolean;
    rep: message;
begin
    …
    a := 1;
    snd(R, a);
if rcv(R,rep,t1) and rep=accept
then
    begin
        snd(R, ack);
        st := rcv(R, b, ∝);
        NORMAL_ACTION
    end
    else
    begin
        snd(R, nack);
        TIMEOUT_ACTION
    end
    …
```

```
var x, y: integer;
    st: boolean;
    rep: message;
begin
    …
    if rcv(S, x, 0) then
    begin
        snd(S, accept);
        st := rcv(S,rep, ∝);
        if rep = ack then
            snd(S, f(x));
        else
        DEFAULT_ACTION
        end
    else
        DEFAULT_ACTION;
    …
end
```

Readers – Writers problem



Readers – Writers problem

```
with ada.text_io, ada.integer_text_io;
use ada.text_io, ada.integer_text_io;
procedure rw is
    n : constant integer := 2;

    task control is
        entry start_read;
        entry stop_read;
        entry start_write;
        entry stop_write;
    end control;

    task body control is
        done : boolean;
        readers : integer range 0 .. n;
        writer : boolean;
```

Readers – Writers problem

```
readers := 0;
writer := false;
loop
        select
                when (not writer) =>
                        accept start_read;
                        Put_Line ("start_read");
                        readers := readers + 1;
                        null;
        or
                accept stop_read;
                Put_Line ("stop_read");
                readers := readers - 1;
                null;
```

Readers – Writers problem

```
                when (not writer) and (readers = 0) =>
                        accept start_write;
                        writer := true;
                        Put_Line ("start_write");
                        null;
        or
                accept stop_write;
                writer := false;
                Put_Line ("stop_write");
                null;
        or
                terminate;
        end select;
        exit when done;
        end loop;
end control;
```

Readers – Writers problem

```
task reader_1 is
end reader_1;
task body reader_1 is
        done : boolean;
        i : integer :=0;
begin
        loop
                control.start_read;
                Put_Line ("reader_1 " & i'Img);
                control.stop_read;
                delay 1.0;
                i := i + 1;
                if i > 10 then
                        done := true;
                end if;
                exit when done;
        end loop;
end reader_1;
```

Readers – Writers problem

```
task writer_1 is
end writer_1;
task body writer_1 is
        done : boolean;
        i : integer :=0;
begin
        loop
                control.start_write;
                Put_Line ("writer_1 " & i'Img);
                control.stop_write;
                delay 1.9;
                i := i + 1;
                if i > 10 then
                        done := true;
                end if;
                exit when done;
        end loop;
end writer_1;
```

Readers – Writers problem

```
    begin
            delay 10.0;
end rw;
```

Producer – Consumer problem

Producer – Consumer

```
with ada.text_io, ada.integer_text_io;
use ada.text_io, ada.integer_text_io;
procedure Producer_Consumer is
    task buffer is
        entry PUT(coin: in integer);
        entry GET(goods: out integer);
    end buffer;
    task body buffer is
    size: constant := 500;
        data: array(1 .. size) of integer;
        indata: integer:=1;
        outdata: integer:= 1;
        num: integer:=0;
```

Producer – Consumer

```
begin
    loop
        select
            when (num < size) =>
                accept PUT(coin: in integer) do
                    data(indata) := coin;
                    indata := (indata) mod size + 1;
                    num:= num + 1;
                end PUT;
                null;
        or
```

Producer – Consumer

```
                    when (num > 1) =>
                              accept GET(goods: out integer) do
                                        goods := data(outdata);
                                        outdata := (outdata) mod size + 1;
                                        num:= num - 1;
                              end GET;
                              null;
          end select;
    end loop;
end buffer;
```

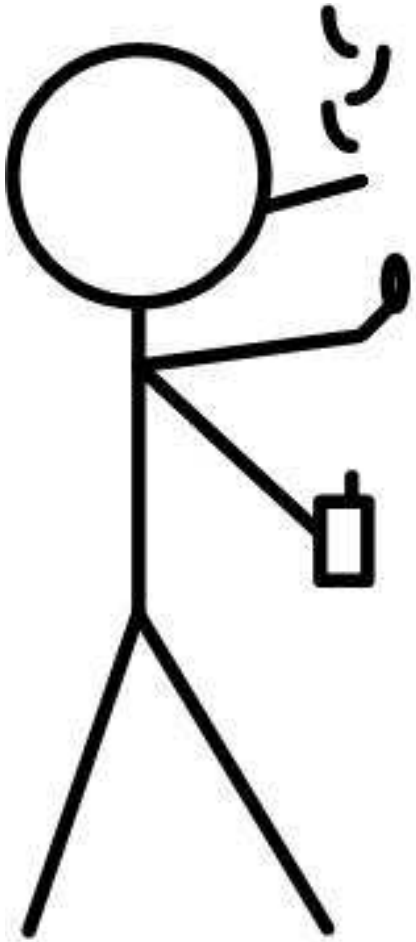
Producer – Consumer

```
task producer1 is
end producer1;
task body producer1 is
i: integer:=0;
begin
        loop
                buffer.PUT(i);
                Put_Line ("buffer.PUT(i)" & i'Img);
                i := i + 1;
                delay 1.0;
        end loop;
end producer1;
```

Producer – Consumer

```
goods: integer := 0;
i: integer :=0;
begin
    loop
        buffer.GET(goods);
        Put_Line ("buffer.GET(goods)" & goods'Img & " " & i'Img);
        i := i + 1;
        delay 1.0;
    end loop;
end Producer_Consumer;
```

Cigarette Smokers' problem



Cigarette Smokers' problem

```
with ada.text_io, ada.integer_text_io,
    ADA.NUMERICS.DISCRETE_RANDOM;
use ada.text_io, ada.integer_text_io;
procedure CS is
    n : constant integer := 2;
    MinNum :constant integer := 1;
    MaxNum :constant integer := 3;

    task Table is
        entry yes_matches;
        entry yes_tobacco;
        entry yes_paper;
        entry goods(inMatches, inTobacco, inPaper: in boolean);
    end Table;
```

Cigarette Smokers' problem

```
task body Table is
    done : boolean;
    matches, tobacco, paper: boolean;
begin
    done := false;
    matches := false;
    tobacco := false;
    paper := false;
    loop
        select
            when (matches AND tobacco) =>
                accept yes_paper do
                    matches := false;
                    tobacco := false;
                    Put_Line ("start yes_paper");
                    null;
```

Cigarette Smokers' problem

```
or
        when (tobacco AND paper) =>
                accept yes_matches do
                        tobacco := false;
                        paper := false;
                        Put_Line ("start yes_matches");
                        null;
                end yes_matches;
or
        when (matches AND paper) =>
                accept yes_tobacco do
                        matches := false;
                        paper := false;
                        Put_Line ("start yes_tobacco");
                        null;
                end yes_tobacco;
```

Cigarette Smokers' problem

```
        or
                accept goods(inMatches, inTobacco, inPaper: in
    boolean) do
                        matches := inMatches;
                        tobacco := inTobacco;
                        paper := inPaper;
                        Put_Line ("goods");
                        null;
                end goods;
        or
                terminate;
        end select;
        exit when done;
    end loop;
end Table;
```

Cigarette Smokers' problem

```
task Agent is
        entry OK;
end Agent;
task body Agent is
subtype myNumber is INTEGER range MinNum .. MaxNum;
package myRandomNumbers is new
ADA.NUMERICS.DISCRETE_RANDOM(myNumber);
        use myRandomNumbers;

        done : boolean;
        i: integer;
        RandomNumber : myNumber;
        Seed : GENERATOR;
begin
```

Cigarette Smokers' problem

```
begin
    RESET(Seed);
    loop
        RandomNumber := RANDOM(Seed);
        i:= RandomNumber;
        Put_Line ("ALIVE");
        if (i = 1) Then
            Table.goods(false, true, true);
            Put_Line ("Agent.goods(false, true, true) " & i'Img);
        elsif (i = 2) Then
            Table.goods(true, false, true);
            Put_Line ("Agent.goods(true, false, true) " & i'Img);
        elsif (i = 3) Then
            Table.goods(true, false, true);
            Put_Line ("Agent.goods(true, true, false) " & i'Img);
        end if;
```

Cigarette Smokers' problem

```
            select
                    accept OK;
                    Put_Line ("OK");
            or
                    terminate;
            end select;

            delay 1.0;
        end loop;
end Agent;
```

Cigarette Smokers' problem

```
task smoker_with_paper is
end smoker_with_paper;
task body smoker_with_paper is
        done : boolean;
        i : integer :=0;
begin
        loop
                Table.yes_paper;
                Put_Line ("yes_paper " & i'Img);
                delay 1.0;
                Agent.OK;
                i := i + 1;
        end loop;
end smoker_with_paper;
```

Cigarette Smokers' problem

```
begin
        delay 1.0;
end CS;
```

Питања?

Захарије Радивојевић
Електротехнички Факултет
Универзитет у Београду
zaki@etf.rs