PROGRAMOWANIE SYSTEMÓW CZASU RZECZYWISTEGO

Dr inż. Bartosz Trybus
D202 C, tel: 865 1685
email: btrybus@prz-rzeszow.pl
www: mail.prz-rzeszow.pl/~btrybus

Dr inż. Sławomir Samolej
D102 C, tel: 865 1766,
email: ssamolej@prz-rzeszow.pl
www: mail.prz-rzeszow.pl/~ssamolej

Program zajęć – Ada

Wykład:

1. **Charakterystyka systemów czasu rzeczywistego,**
2. **Język Ada2005 – narzędzia programowe, struktura programu, jednostki programowe, typy danych, instrukcje, podprogramy,**
3. **Język Ada2005 – pakiety, programowanie współbieżne – zadania,**
4. **Język Ada2005 – programowanie współbieżne - synchronizacja czasowa, komunikacja między zadaniami,**
5. **Język Ada2005 – priorytety i szeregowanie,**
6. **Język Ada2005 – wyjątki i przerwania, odporność na błędy**
7. **Projekt przykładowego systemu czasu rzeczywistego, zaawansowane konstrukcje językowe.**

Laboratorium:

1. **Język Ada2005 – typy danych, instrukcje, podprogramy,**
2. **Język Ada2005 – pakiety, programowanie współbieżne, synchronizacja czasowa, komunikacja,**
3. **Język Ada2005 –priorytety i szeregowanie, projekt prostego systemu wbudowanego.**

Literatura – Ada

- G. Motet, T. Szmuc,
  Programowanie systemów czasu rzeczywistego z zastosowaniem języka Ada, Wydawnictwa AGH 2002.
- Z. Huzar, Z. Fryźlewicz, I. Dubielewicz, B. Hnatk,
  Ada 95, HELION 1998.
- Barnes J.,
  Programming in Ada2005, Addison Wesley 2006.
- Burns A., Wellings A.,
  Concurrency in Ada, Cambridge University Press 1998.
- Burns A, Wellings A.,
  Real-Time Systems and Programming Languages, third edition, Pearson Education Limited 2001.
- Butazzo G. C.
  Hard Real-Time Computing Systems, Predictable Scheduling Algorithms and Applications, Kluwer Academic Publishers 1997.
- Odnośniki, strona mail.prz-rzeszow.pl/~ssamolej

Warunki uzyskania zaliczenia

- **Uczestnictwo w zajęciach laboratoryjnych.**
- **Przedłożenie projektu prostego systemu czasu rzeczywistego.**

Spektakularne porażki systemów czasu rzeczywistego

- czego będziemy się uczyć unikać

- **Przerwane odliczanie przed pierwszym startem wahadłowca.**
- **Pomyłka wyrzutni Patriot 25 lutego 1991.**
- **Zablokowana odpalona rakieta w samolocie F18**

Definicje

System czasu rzeczywistego jest to system komputerowy, w którym obliczenia są wykonywane współbieżnie z procesem zewnętrznym (otoczenie) w celu sterowania, nadzorowania lub terminowego reagowania na zdarzenia występujące w tym procesie (otoczeniu).

- **Na poprawność systemu składa się zarówno wyliczanie logicznych wyników jak i dostarczanie ich w określonym czasie.**
- **Nieterminowe wyliczenie odpowiedzi jest błędem w równym stopniu co podanie odpowiedzi błędnej.**

Jeśli komputer jest częścią większej instalacji przemysłowej, to często jest jest nazywany wbudowanym systemem komputerowym (ang. embedded computer system).

99% procesorów wykorzystywanych jest w aplikacjach wbudowanych.

Spotykane pojęcia

- **Systemy o twardych wymaganiach czasowych (ang. Hard Real-Time Systems)**
  Systemy, gdzie ograniczenia czasowe muszą być zawsze spełnione, np. system kontroli lotu myśliwca.
- **Systemy o miękkich wymaganiach czasowych (ang. Soft Real-Time Systems)**
  Systemy, gdzie ograniczenia czasowe są istotne, ale uznaje się, że działają poprawnie, jeśli od czasu do czasu nastąpi przekroczenie ograniczeń, np. system akwizycji danych.
- **Systemy o solidnych wymaganiach czasowych(ang. Firm Real-Time Systems)**
  Systemy o miękkich wymaganiach czasowych, w których spóźniona odpowiedź jest traktowana jako błędna.

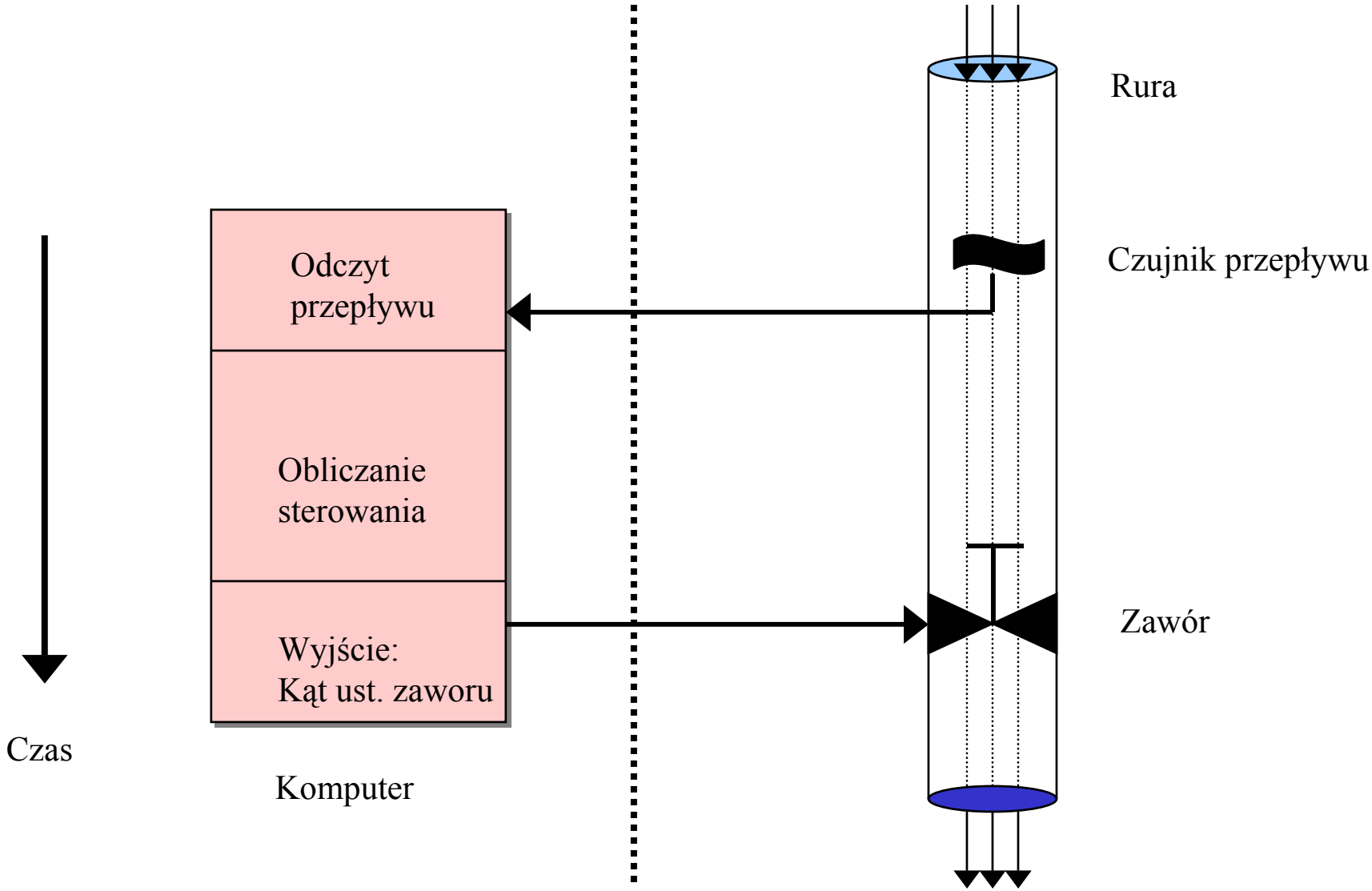
Pojedynczy system czasu rzeczywistego posiada zwykle podsystemy o twardych i miękkich wymaganiach czasowych. Definiowane są również funkcje kosztu przekroczenia ograniczeń czasowych.

Spotykane pojęcia

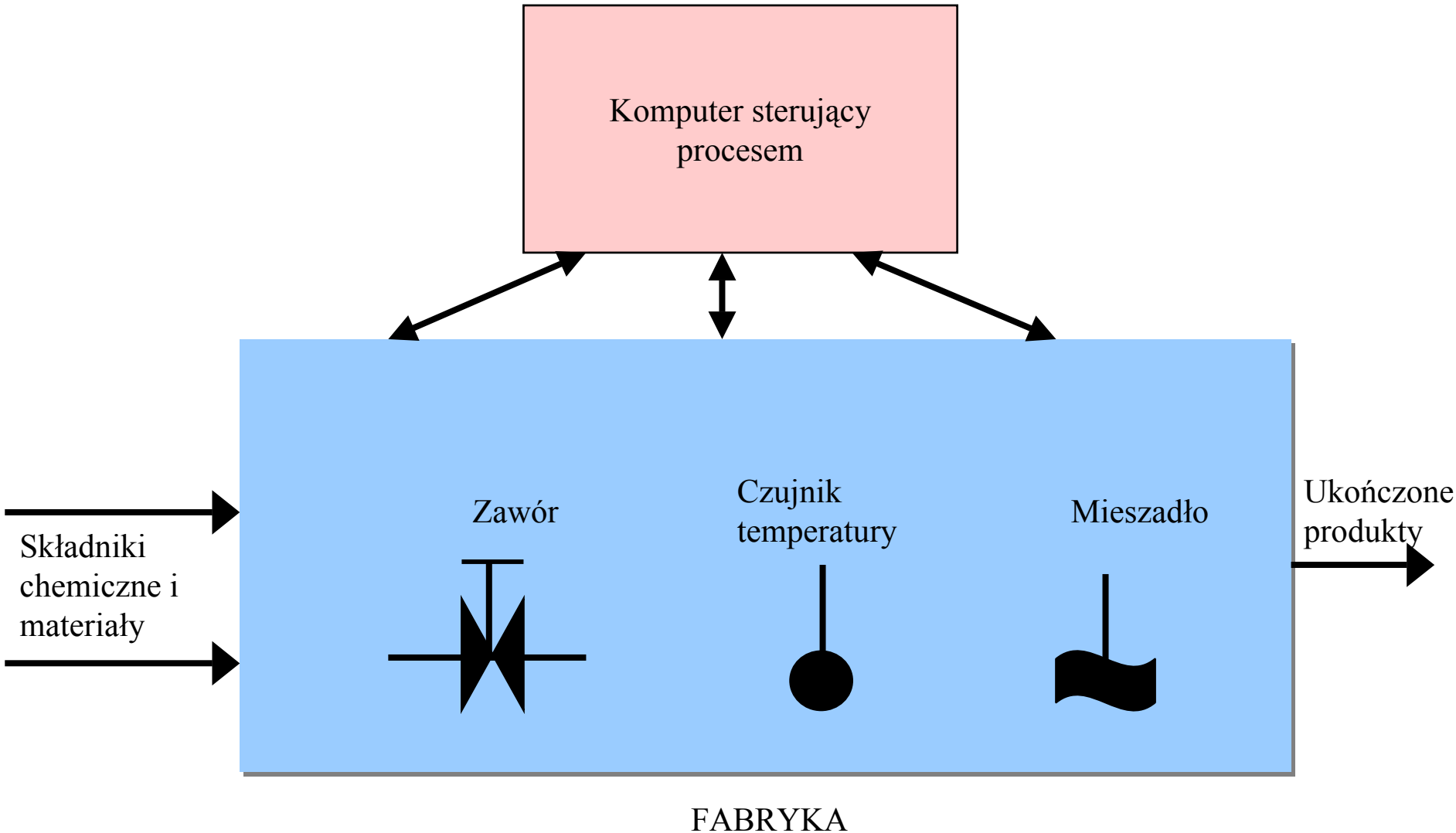
- **Systemy wysokozintegrowane (ang. High Integity Systems)**
  Systemy, w których oprogramowanie steruje pewnymi procesami mającymi istotny wpływ na ludzi, środowisko, organizacje i społeczeństwo.
  Systemy wysokozintegowane dzieli się na:
  - **Systemy krytyczne ze względu na bezpieczeństwo (ang. Safety Critical Systems), gdzie wynik działania systemu ma bezpośredni wpływ na życie, zdrowie człowieka lub na stan środowiska, np. systemy sterowania samolotu, ruchem pociągów, oprogramowanie samochodów i innych środków transportu, zabezpieczenia systemów energetycznych, urządzenia medyczne.**
  - **Systemy krytyczne ze względu napełniona misję (ang. Mission Critical Systems), gdzie poprawność działania oprogramowania ma poważny wpływ na działanie produkcji, instytucji, organizacji**

Szereg systemów krytycznych ze względu na bezpieczeństwo jest systemami czasu rzeczywistego.

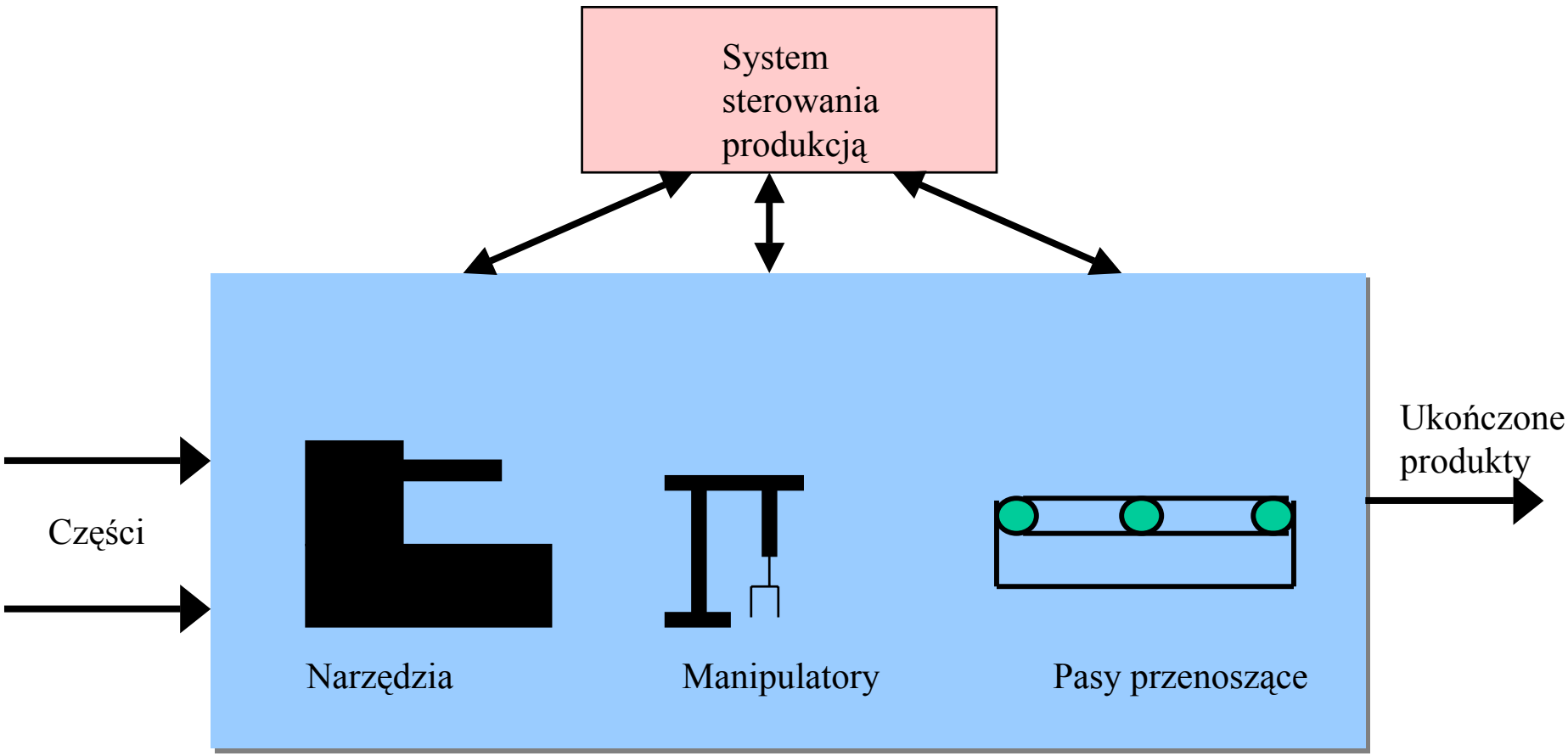
Przykłady systemu czasu rzeczywistego - system sterowania przepływem



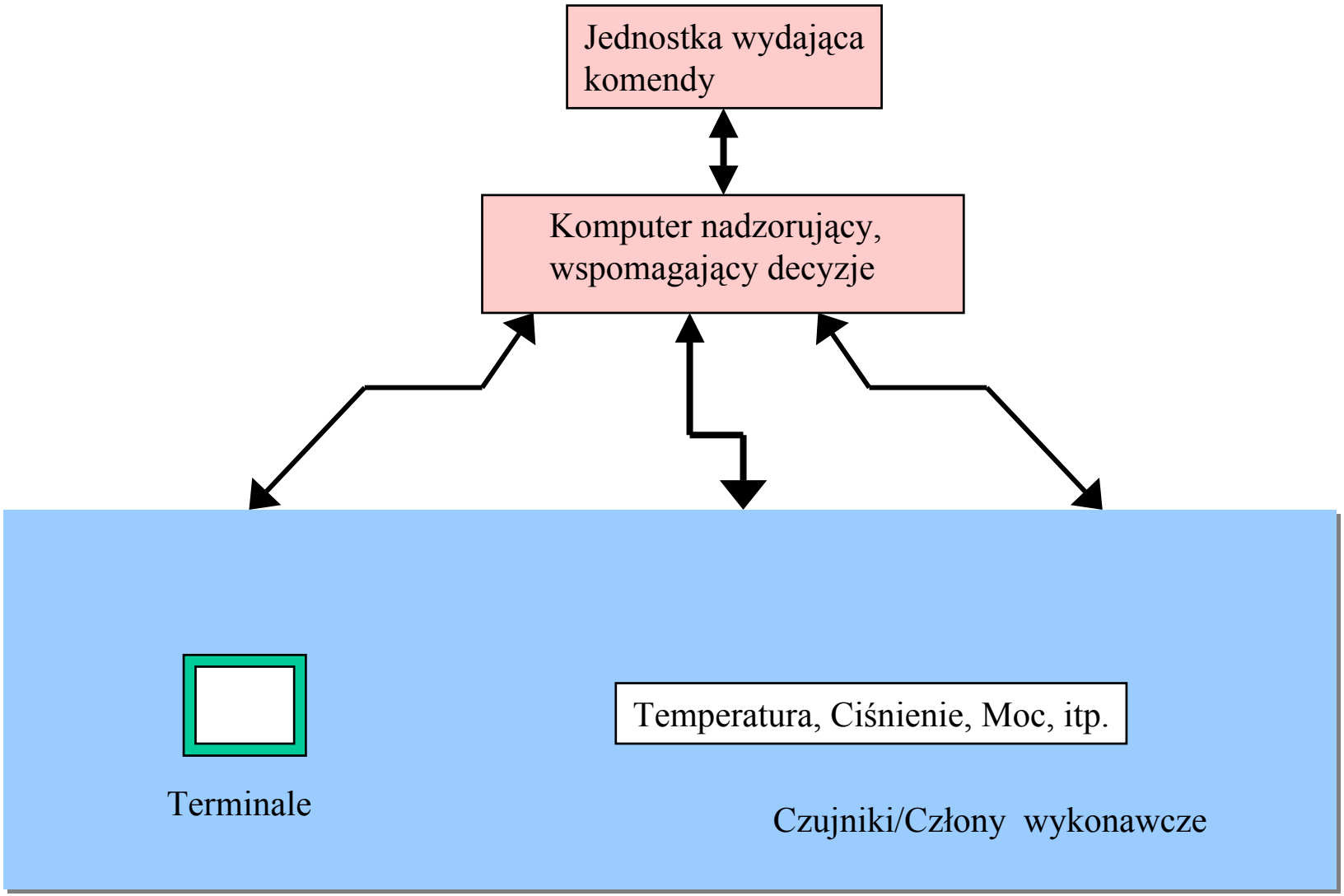
Przykłady systemu czasu rzeczywistego – system sterowania procesem



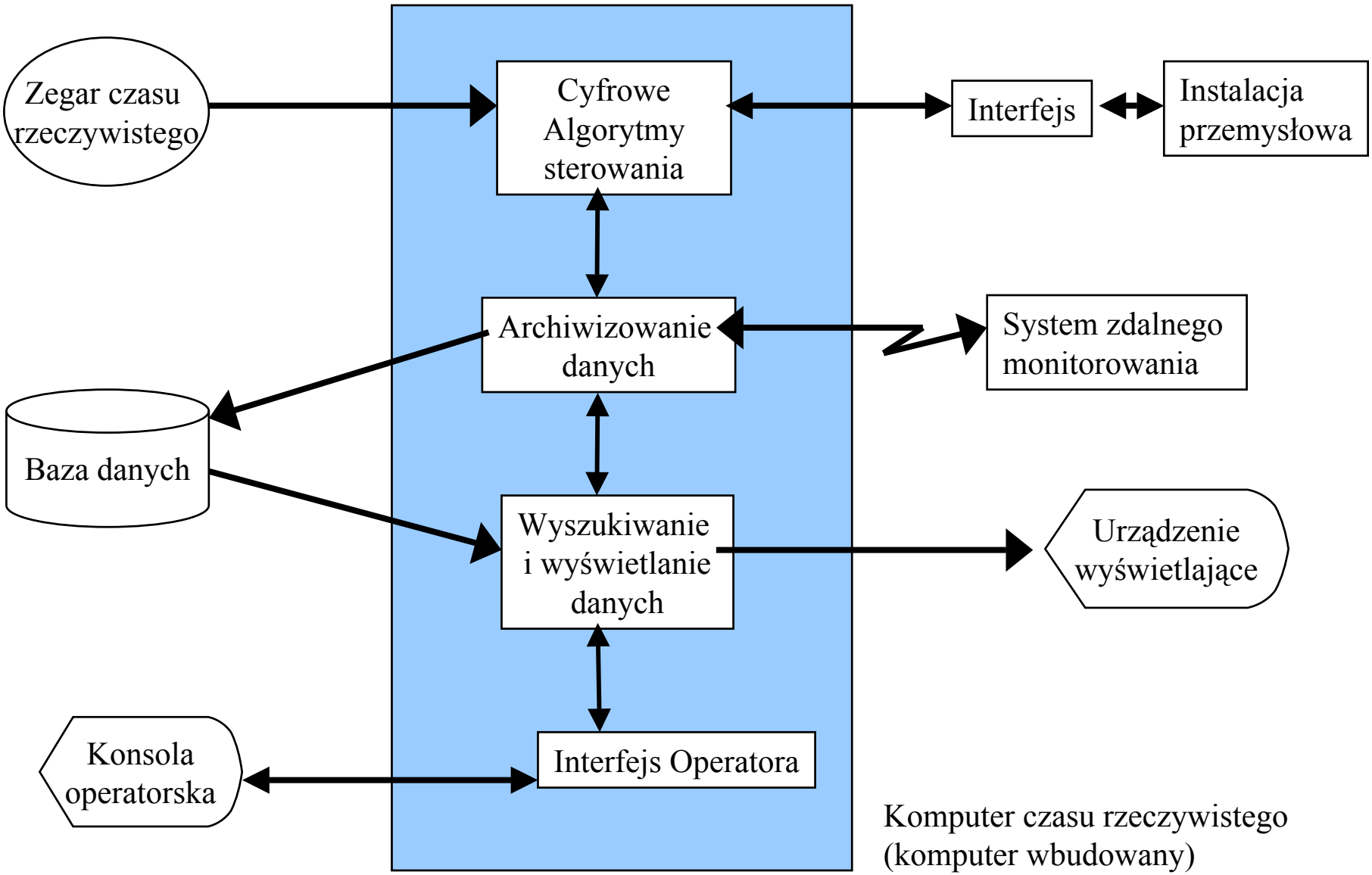
Przykłady systemu czasu rzeczywistego – system sterowania produkcją



Przykłady systemu czasu rzeczywistego – system „command and control”



Systemu czasu rzeczywistego – uogólniony schemat



Komputer czasu rzeczywistego
(komputer wbudowany)

Przykłady aplikacji wymagających oprogramowania czasu rzeczywistego

- **Sterowanie zakładami chemicznymi i elektrowniami nuklearnymi**
- **Sterowanie złożonymi procesami produkcyjnymi**
- **Zarządzaniem ruchem kolejowym**
- **Aplikacje stosowane w pojazdach samochodowych (ABS, EBD)**
- **Autopiloty i inne podsystemy wspomagające pracę samolotu**
- **Monitorowanie i akwizycja danych**
- **Telekomunikacja**
- **Automatyka przemysłowa i robotyka**
- **Systemy wojskowe**
- **Misje kosmiczne**
- **Wirtualna rzeczywistość**

Cechy systemu czasu rzeczywistego

- **Ciągłość działania**
  Po wdrożeniu system zwykle pracuje bez przerw, stale monitorując, nadzorując i podejmując decyzje.
- **Zależność od otoczenia**
  Zwykle system rozpatrywany jest z otoczeniem, nieraz złożonym, ale zwykle statycznym, co pozwala na ograniczenie dynamicznych struktur danych (np. z założenia określa się ilość obsługiwanych urządzeń, kanałów wejściowych itp.).
- **Współbieżność**
  Ponieważ otoczenie składa się zwykle z wielu współbieżnych procesów, oprogramowanie tworzy się również jako zbiór współbieżnych zadań, równolegle reagujących na zdarzenia w otoczeniu.
- **Przewidywalność**
  Pomimo współbieżnego, generującego zdarzenia w przypadkowych momentach otoczenia, system, także współbieżny, musi w sposób przewidywalny generować odpowiedzi.
- **Punktualność**
  Oczekuje się, że odpowiedzi systemu obliczane będą zgodnie z ustalonymi ograniczeniami czasowymi.

Metody programowania systemów czasu rzeczywistego

1. **Tworzenie aplikacji bezpośrednio zarządzających elementami systemu mikrokomputerowego (por. Mikroinformatyka)**
2. **Programowanie sterowników swobodnie programowalnych**
3. **Zastosowanie języków programowania systemów czasu rzeczywistego (Ada2005, Occam, Pearl, Real-Time Java).**
   - I część przedmiotu (dr inż. S. Samolej)
4. **Zastosowanie wbudowanych funkcji systemów operacyjnych czasu rzeczywistego (język C/ C++).**
   - II część przedmiotu (dr inż. B. Trybus)

Wybrane najpopularniejsze systemy operacyjne czasu rzeczywistego

- **VxWorks**
- **XP Embedded**
- **Windows CE**
- **QNX**
- **RT Linux, Linux/RT, Red Hat Linux i inne**

Cechy języków programowania systemów czasu rzeczywistego

- **Możliwość zdefiniowania spójnych części, realizujących obsługę zdarzeń pojawiających się w otoczeniu – zadań (aktywowanych zdarzeniami lub upływem czasu).**
- **Konieczność istnienia mechanizmów synchronizacji i wymiany informacji pomiędzy zadaniami.**
- **Konieczność określenia dla zadań priorytetu oraz możliwość wywłaszczania zadania.**
- **Możliwość uzależnienia działania systemu od czasu i innych zdarzeń zachodzących w otoczeniu.**
- **Możliwość definiowania procedur obsługi dla nietypowych (np. procesowych) wejść i wyjść.**
- **Istnienie mechanizmów umożliwiających konstruowanie oprogramowania o podwyższonej niezawodności (np. obsługa wyjątków).**

Historia języka Ada

- **Lata siedemdziesiąte – Departament Obrony Stanów Zjednoczonych rozpoczyna działania w kierunku zdefiniowania nowego języka programowania do tworzenia oprogramowania złożonych systemów wbudowanych (sterowanie procesami chemicznymi i rakietami)**
- **1980 – Uniwersytet w Nowym Jorku – pierwszy interpreter**
- **1983 – Ada83 – standard ANSI**
- **1987 – standard ISO, 1988 – poprawiny standard**
- **1990 – nowe wymagania dla języka**
- **1995 – standaryzacja ISO - Ada95**
- **2001 – korekta standardu ISO – Ada95**
- **2005 – propozycja poprawionego języka o nazwie Ada 2005, złożona do ISO, spodziewane zatwierdzenie standardu: przełom 2006/2007.**

- **Nazwa języka pochodzi od Ady Augusty Byron (1815-1852), księżnej Lovelace, córki lorda Byrona, asystentki Karola Babage – konstruktora mechanicznej maszyny analitycznej, uważanej za pierwszą osobę programującą w świecie.**

Znane aplikacje wytworzone z zastosowaniem języka Ada (http://www.adahome.com/Ammo/Success/)

- Systemy transportowe:
  - Sterowanie i symulacja szybkiej kolei francuskiej (TGV)
  - Sterowanie metrem w Paryżu, Kairze, Kalkucie
  - Sterowanie koleją miejską w Hong Kongu
- Systemy lotnicze:
  - Myśliwiec Eutrfighter Typhoon
  - Europejski system kontroli powietrznej
  - Sterowanie zasilaniem i hamulcami w samolocie Boeing 777
  - System ostrzegania w samolocie Airbus 340
- Systemy telekomunikacji
  - Systemy GPS
  - Systemy telefonii komórkowej
  - Sterowanie radioteleskopami
- Rozrywka
  - Systemy do edycji video i wspomagania prowadzenia programów informacyjnych

Struktura definicji języka Ada 2005

- **język składa się z:**
  - **Części rdzeniowej (musi być w każdym kompilatorze)**
  - **Zbioru opcjonalnych aneksów w tym:**
    - **System Programming**
    - **Real-Time Programming**
    - **High Integity Systems**

Jednostki programowe języka Ada

- **Podprogramy (subprograms) – funkcje i procedury**
- **Pakiety (packages) – kolekcje definicji (dane + inne jednostki programowe)**
- **Zadania (tasks) – opisujące proces wykonywane współbieznie oraz komunikację między zadaniami**
- **Obiekty chronione (protected objects) – implementujące pamięć dzieloną, umożliwiające definiowanie komunikacji między zadaniami w trybie dzielonej pamięci**
- **Jednostki rodzajowe (generic units) – umożliwiające definiowanie uogólnionych jednostek programowych niezależnych od typów danych**

Ada – przykład wstępny

```
// C/C++:
#include <stdio.h>

void Print_Text(void)
{ char ch;

puts("Druk wprowadzonych
    znaków");
putchar('\n');
while(1)
    {   ch=getchar();
        if(ch=='?') break;
        putchar(ch);
    }
putchar('\n');
puts("Koniec programu");
}
void main(void)
{Print_Text();}
```

```
-- Ada:
With Text_IO; Use  Text_IO;

Procedure Print_Text is
    ch: Character;
Begin
    Put("Druk wprowadzonych
    znaków");
    New_Line(2);
    loop
        Get(ch);
        exit when ch='?';
        Put(ch);
    end loop;
    New_Line;
    Put("Koniec programu");
End Print_Text;
```

Ada - operatory

Operator	C/C++	Ada
Przypisanie	=	:=
Porównanie	==	=
Nierówność	!=	/=
	+=, -=. *=, /=, |=, @=	brak
Reszta z dzielenia	%	mod / rem
Wartość bezwzględna	Brak	abs
Potęgowanie	Brak	**
Zakres	Brak	range

Ada – typy danych

- **Definiowanie zmiennych:**

```
// C/C++:
int i;
int a, b, c;
int j = 0;
int k, l = 1;
```

```
-- Ada:
i : Integer;
a, b, c : Integer;
j : Integer := 0;
k, l : Integer := 1;
```

- **Definiowanie stałych:**

```
// C/C++:
const int
    days_per_week = 7;
```

```
-- Ada:
days_per_week :
    constant Integer := 7;
days_per_week :
    constant := 7;
```

Ada – typy danych

- **Definiowanie nowych typów i podtypów:**

```
// C/C++:
typedef int INT;
INT a;
int b;
a = b; //działa
```

```
-- Ada:
type INT is new Integer;
a : INT;
b : Integer;
a := b;  -- nie działa!
         -- Ścisła typizacja!

-- Rozwiązanie problemu:
subtype INT is Integer;
a : INT;
b : Integer;
a := b; -- działa!
        -- INT jest podtypem
```

Ada – typy danych

- **Typy proste: całkowity i znakowy**

```
-- Typ całkowity:
Integer -- całkowity ze znakiem (musi być w każdym
        -- kompilatorze)
-- Dodatkowo (nieobowiązkowe dla kompilatorów):
Long_Integer, Short_Integer, Long_Long_Integer
-- Brak jest zdefiniowanego typu Unsigned, można go
-- zdefiniować z zastosowaniem operatora zakresu range
-- lub zastosować pakiet System.Unsigned_Types

-- Typ całkowity resztowy:
type BYTE_8 is mod 8;        -- zakres 0..7  (cyklicznie)
type BYTE_16 is mod 16;      -- zakres 0..15 (cyklicznie)

-- Typ znakowy:
Character

-- Typ logiczny:
Boolean -- wartości: (False, True)
        -- operatory: not, and, or, xor
        -- koniecznie stosować NAWIASY! (ten sam priorytet)
```

Ada – typy danych

- **Łańcuchy tekstowe**

```
-- - Istnieje predefiniowany typ łańcuchowy: String
-- - Wprowadzono operator łączenia łańcuchów: &
-- - Łańcuch jest rozumiany jako tablica znaków
-- - Łańcuchy muszą mieć zawsze zdefiniowaną długość!
-- Przykładowe deklaracje:

type A_Record is record
   illegal : String;        -- błąd! Niezdefiniowana długość
   legal : String(1 .. 20);
end record;

procedure check(legal : in String);
                -- w procedurach i funkcjach wystarczy podać
                -- tylko typ danych bez długości

-- - Rozmiar łańcucha musi się zaczynać od 1!
```

Ada – typy danych

- **Typy stało i zmiennoprzecinkowe**

```
-- standardowy zmiennopozycyjny:
type FloatingPoint1 is new Float;

-- typ zmiennopozycyjny, w którym
-- można określić precyzję i zakres liczb danego typu:
type FloatingPoint2 is digits 6 [range 17.5 .. 32.7];

-- typ stałopozycyjny zwykły:
type Fixed is delta 0.1 range -1.0 .. 1.0;
    -- od –1.0 do 1.0 z dokładnością 0.1

-- typ stałopozycyjny dziesiętny:
type Decimal is delta 0.01 [digits 10];
    -- 99,999,999.99 ... +99,999,999.99
```

Ada – typy danych

- **Typ wyliczeniowy**

```
type Boolean is (FALSE, TRUE);
type Hours is new Integer range 1 .. 12;

type All_Days is (Monday, Tuesday, Wednesday, Thursday,
   Friday, Saturday, Sunday);
subtype Week_Days is All_Days range Monday .. Friday;
subtype Weekend is All_Days range Saturday .. Sunday;
```

- **Atrybuty typów wyliczeniowych**

```
All_Days'Succ(Monday) = Tuesday
All_Days'Pred(Tuesday) = Monday
All_Days'Val(0) = Monday
All_Days'First = Monday
All_Days'Val(2) = Wednesday
All_Days'Last = Sunday
All_Days'Succ(All_Days'Pred(Tuesday)) = Tuesday
```

Ada – typy danych

- **Tablice**

```
//C:
char name[31];
int track[3];
int dbla[3][10];
int init[3] = { 0, 1, 2 };
typedef char[31] name_type;
track[2] = 1;
dbla[0][3] = 2;

-- Ada
Name : array (0 .. 30) of Character; -- lub:
Name : String (1 .. 31);
Track : array (0 .. 2) of Integer;
DblA : array (0 .. 2) of array (0 .. 9) of Integer; -- lub:
DblA : array (0 .. 2,0 .. 9) of Integer;
Init : array (0 .. 2) of Integer := (0, 1, 2);
type Name_Type is array (0 .. 30) of Character;
track(2) := 1;
dbla(0,3) := 2;
```

Ada – typy danych

- **Tablice o nieograniczonym rozmiarze**

```
-- Ada
type Tablica is array (Integer range <>) of Integer;

t1: Tablica (0..5) := (1,-3,4,-5,3,2);
T2: Tablica (0..9);

procedure kopiuj(
                    dane_zrodlowe: in out Tablica;
                    dane_docelowe: in out Tablica
                );
```

Ada – typy danych

- **Rekordy**

```
//C:
struct _device {
    int major_number;
    int minor_number;
    char name[20]; };
typedef struct _device Device;
Device lp1 = {1, 2, "lp1"};
lp1.major_number = 12;
--Ada:
type struct_device is record
    major_number : Integer;
    minor_number : Integer;
    name : String(1 .. 19);
end record;
type Device is new struct_device;

lp1 : Device := (1, 2, "lp1");
lp2 : Device :=     (major_number => 1,
                     minor_number => 3, name => "lp2");
tmp : Device := (major_number => 255, name => "tmp");
lp1.major_number := 12;
```

Ada – instrukcje

- **Instrukcja złożona**

```
// C/C++:
{
declarations
statements
}
```

```
-- Ada:
declare
    declarations
begin
    statements
end;
```

- **Instrukcja *if***

```
// C/C++:
if (expression)
{ statement
}
else
{ statement
}
```

```
-- Ada:
if expression then
    statement
elsif expression then
    statement
else
    statement
end if;
```

Ada – instrukcje

- **Instrukcja *switch***

```
// C/C++:
switch (expression)
{   case value: statement
    default: statement
}
```

```
-- Ada:
case expression is
   when value => statement
   when others => statement
end case;
```

- **Zastosowanie zasięgu i zbioru wartości w instrukcji *switch***

```
// C/C++:
switch (integer_value)
{ case 1: case 2:
  case 3: case 4:
       value_ok = 1; break;
  case 5: case 6: case 7:
            break;
 }
```

```
-- Ada:
case integer_value is
   when 1 .. 4
      => value_ok := 1;
   when 5 | 6 | 7
      => null;
end case;
```

Ada – instrukcje

- **Instrukcje pętli**

```
-- Ada – instrukcja pętli
loop
    statement
    [exit when expression]
end loop;
```

- **Pętla *while***

```
// C/C++:
while (expression)
{
statement
}
```

```
-- Ada:
while expression
loop
    statement
end loop;
```

Ada – instrukcje

- **Pętla *for***

```
// C/C++:
for (init-statement ;
     expression-1 ;
     loop-statement)
{
statement
}
```

```
-- Ada:
for ident in [reverse] range
loop
   statement
end loop;
```

```
-- Przykłady:
for i in 1 .. 10
loop
   null;
end loop;
-----------------------------
for i in reverse 1 .. 10
loop
   null;
end loop;
```

Ada – instrukcje

- **Odpowiednik pętli *do*, odpowiednik instrukcji *break* (brak jest odpowiednika instrukcji *continue*)**

```
// C/C++:
do
{
statement
}
while (expression)
//////////////////////////
while (expression)
{
    if (expression2)
    { break; }
}
```

```
-- Ada:
loop
    statement
    exit when expression;
end loop;

----------------------------
while expression
loop
    if expression2 then exit;
    end if;
end loop;
```

Ada – instrukcje

- **Instrukcja *return***

```
// C/C++:
return value;
```

```
-- Ada:
return value;
```

- **Instrukcja *goto***

```
// C/C++:
label:
    goto label;
```

```
-- Ada:
<<label>>
     goto label;
```

Ada – instrukcje

- **Przechwytywanie wyjątków (exceptions)**

```
// C++:
try { statement1
} catch (declaration) {
     statement2
}


// Przykład
try
{ function_call();
} catch (const char*
        string_exception)
    { if
    (!strcmp(string_exception
        ,"the_one_we_want"))
      {handle_it();}
      else {throw;}
    }
catch (...) {throw;}
```

```
-- Ada:
begin statement1
    exception when ident =>
        statement2
    when others =>
        statement2
end;
-- Przykład:
begin
    function_call;
exception
when the_one_we_want =>
                        handle_it;
when others => raise;
end;
```

Ada – podprogramy

- **Funkcje:**

```
// C/C++:
return_type func_name(parameters);

return_type func_name(parameters)
{
declarations
statement
}

-- Ada:
function func_name(parameters) return return_type;

function func_name(parameters) return return_type is
   declarations
begin
   statement
end func_name;
```

Ada – podprogramy

- **Przykład funkcji:**

```
function add1 (dana: Integer) return Integer is
dana1 : Integer;
begin
    dana1 := dana + 2;
    return dana1;
end add1;
-- Wywołanie funkcji:
procedure wypisz1 is
        V :  array (Integer range 1..10)
              of Integer := (1..4 => 1, others => 10);
        nr : Integer;
begin
    nr := 1;
    V(nr) = add1(V(nr));
   Put(V(nr));
end wypisz1;
```

Ada – podprogramy

- **Procedury:**

```
// C/C++:
void func_name(parameters);
```

```
-- Ada:
procedure
    func_name(parameters);
```

- **Przekazywanie parametrów do funkcji/procedur:**

```
// C/C++:


void func1(int by_value);

void func2(int* by_address);

void func3(int& by_reference);
                // tylko C++
```

```
-- Ada:
type int is new Integer;
type int_star is access int;

procedure func1(by_value:
                        in int);
procedure func2(by_address:
                  in out int_star);
procedure func3(by_reference:
                      in out int);
procedure func4(ret_val:
                       out int);
```

Ada – podprogramy

- **Przykład procedury:**

```
procedure policz_kwadrat is
      A1, B1, C1, X, Y : Float;
      Status : Boolean;
-- Procedura zdefiniowana wewnątrz procedury!:
procedure Dwu_Kwadrat( A, B, C: in Float; X1, X2: out Float;
                       OK: out Boolean) is
   D: Float;
   begin
      D := B**2-4.0*A*C;
      if D < 0.0 or A = 0.0 then
         OK := False; X1 := 0.0; X2 := 0.0; return;
      end if;
      X1 := (-B+Sqrt(D)) / (2.0*A);
      X2 := (-B-Sqrt(D)) / (2.0*A); OK := True;
   end Dwu_Kwadrat;
-- Wywołanie procedury:
begin
      A1 := 10.0; B1 := 200.0; C1 := 30.0;
      Dwu_Kwadrat(A1, B1, C1, X , Y, Status);
end policz_kwadrat;
```

Ada – przykładowe programy

```
-- Obliczneie wartości liczby e=1 + 1/1! + 1/2! + 1/3!...
with Text_IO; use Text_IO;
with ada.integer_text_io; use ada.integer_text_io;
with ada.float_text_io;   use ada.float_text_io;
with Ada.Numerics.Elementary_Functions;
use Ada.Numerics.Elementary_Functions;
procedure count_E is
    E: Float :=1.0;
    I: Integer :=0;
    Term: Float:=1.0;
    N: constant Integer := 100;
begin
    loop
        exit when I=N; -- to samo co: if I=N then exit; end if;
        I:=I+1;
        Term:=Term/Float(I);
        E:=E+Term;
    end loop;
    Put_Line("Wyliczona weartosc e:");
    Put(E);
end Count_E;
```

Ada – przykładowe programy

```
-- Obliczneie wartości liczby e - warianty pętli:


-- While:
    while I /= N loop
        I:=I+1;
        Term:=Term/Float(I);
        E:=E+Term;
    end loop;

-- For:
    for I in 1..N loop
        Term:=Term/Float(I);
        E:=E+Term;
    end loop;
```

Ada – przykładowe programy

```
-- Tablice jednowymiarowe:
procedure arrays1 is
A : array(Integer range 1..6) of Float;
type Vector_6 is array (1..6) of Float;
A1, A2 : Vector_6;
begin
    for I in 1..6 loop
       A(I):=Float(I+10);
    end loop;
    Put(A'First); Put(A'Last); Put(A'Length);
    A1:=Vector_6(A);
    New_Line;
    for I in A1‘Range loop
       put(A1(i));
    end loop;
    New_Line;
    A2 := (1=>5.0,2..4=>3.0,others=>20.0);
    for I in A2‘Range loop
       put(A2(i));
    end loop;
end arrays1;
```

Ada – przykładowe programy

```
-- Tablice dwuwymiarowe:
procedure arrays2 is
AA: array(0..1,0..2) of float
    :=   (     (1.0,2.0,3.0),
               (4.0,5.0,6.0));
begin
    AA:=       (0 => (0 => 50.0, others => 20.0),
                1 => (2 =>100.0, others => 10.0)
               );
    for I in AA'Range(1) loop
        for j in AA'Range(2) loop
               Put(AA(I,J));
        end loop;
        New_Line;
    end loop;
    Put(AA'First(1)); Put(AA'Last(2)); Put(AA'Length(2));
end arrays2;
```

Ada – przykładowe programy

```
-- Rekordy:
procedure Record1 is
type Month_Name is (Jan, Feb, Mar, Apr, May, Jun,
                    Jul, Aug, Sep, Oct, Nov, Dec);
type Date is
    record
        Day: Integer range 1..31;
        Month: Month_Name;
        Year: Integer;
    end record;
D,E: Date :=(4, Jan, 1766);
begin
    Put(D.Day); Put(Month_Name'Pos(D.Month) + 1); Put(D.Year);
    New_Line;
    E:=D;
    Put(E.Day); Put(Month_Name'Pos(E.Month) + 1); Put(E.Year);
    New_Line;
    E:=(8,Year=>1990,Month=>Dec);
    Put(E.Day); Put(Month_Name'Pos(E.Month) + 1); Put(E.Year);
    New_Line;
end Record1;
```