Latex
Algorytmy i Listingi

dr hab. Bożena Woźna-Szcześniak

Akademia im. Jan Długosza

bwozna@gmail.com

Laboratorium 9

Środowisko `verbatim`

Domyślnym środowiskiem do wyświetlania kodu w LaTeX jest środowisko `verbatim`, które generuje wyjście za pomocą czcionki o stałej szerokości.

Kod

```
\begin{verbatim}
void SortowanieBabelkowe( int tab[], int size )
{
  for( int i = 0; i < size; i++ )
   {
    for( int j = 0; j < size - 1; j++ )
    {
      if( tab[ j ] > tab[ j + 1 ] )
        swap( tab[ j ], tab[ j + 1 ] );
    }
  }
}
\end{verbatim}
```

Środowisko `verbatim`

Wykonanie

```
void SortowanieBabelkowe( int tab[], int size )
{
    for( int i = 0; i < size; i++ )
    {
        for( int j = 0; j < size - 1; j++ )
        {
            if( tab[ j ] > tab[ j + 1 ] )
                swap( tab[ j ], tab[ j + 1 ] );
        }
    }
}
```

Środowisko `verbatim`

Drugim domyślnym środowiskiem do wyświetlania kodu w LaTeX jest środowisko `verbatim*`, które generuje wyjście za pomocą czcionki o stałej szerokości i specjalnie oznacza białe znaki.

Kod

```
\begin{verbatim*}
void SortowanieBabelkowe( int tab[], int size )
{
  for( int i = 0; i < size; i++ )
   {
    for( int j = 0; j < size - 1; j++ )
    {
      if( tab[ j ] > tab[ j + 1 ] )
        swap( tab[ j ], tab[ j + 1 ] );
    }
  }
}
\end{verbatim*}
```

Środowisko `verbatim`

Wykonanie

```
void␣SortowanieBabelkowe(␣int␣tab[],␣int␣size␣)
{
␣␣␣␣for(␣int␣i␣=␣0;␣i␣<␣size;␣i++␣)
␣␣␣␣{
␣␣␣␣␣␣␣␣for(␣int␣j␣=␣0;␣j␣<␣size␣-␣1;␣j++␣)
␣␣␣␣␣␣␣␣{
␣␣␣␣␣␣␣␣␣␣␣␣if(␣tab[␣j␣]␣>␣tab[␣j␣+␣1␣]␣)
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣swap(␣tab[␣j␣],␣tab[␣j␣+␣1␣]␣);
␣␣␣␣␣␣␣␣}
␣␣␣␣}
}
```

Listowanie programów

Do listowania programów służy pakiet **listings** oraz środowisko `lstlisting`

Kod

```
\begin{lstlisting}[language=Python]
n = int(input("Podaj liczbe naturalna: "))
s = 0
for j in range(1, n + 1):
    print(" poziom ", j, " : ", s, "+", j, " = ")
    s = s + j
    print(s)
print("Suma liczb od 1 do", n, "wynosi", s)
\end{lstlisting}
```

Listowanie programów

Wykonanie

```
n␣=␣int(input("Podaj␣liczbe␣naturalna:␣"))
s␣=␣0
for␣j␣in␣range(1,␣n␣+␣1):
␣␣␣␣print("␣poziom␣",j,␣"␣:␣",␣s,␣"+",␣j,␣"␣=␣")
␣␣␣␣s␣=␣s␣+␣j
␣␣␣␣print(s)
print("Suma␣liczb␣od␣1␣do",␣n,␣"wynosi",␣s)
```

Importowanie do listingu danych z pliku

Kod

```
\lstinputlisting[language=Python]{z2.py}
```

Importowanie do listingu danych z pliku

Wykonanie

```
import␣math

def␣main():
␣␣␣n␣=␣int(input("Podaj␣ilosc␣liczb:␣"))
␣␣␣b␣=␣float(input("Podaj␣kolejna␣liczbe␣rzeczywista:␣"))
␣␣␣for␣j␣in␣range(0,␣n-1):
␣␣␣␣␣␣a␣=␣float(input("Podaj␣kolejna␣liczbe␣rzeczywista:␣"))
␣␣␣␣␣␣print(a)
␣␣␣print(b)

main()
```

Importowanie do listingu danych z pliku

Wyświetlanie kodu od linii 3 do linii 9.

Kod

```
\lstinputlisting[language=Python,
 firstline=3, lastline=9]{z2.py}
```

Importowanie do listingu danych z pliku

Wykonanie

```
def main():
    n = int(input("Podaj ilosc liczb: "))
    b = float(input("Podaj kolejna liczbe rzeczywista: "))
    for j in range(0, n-1):
        a = float(input("Podaj kolejna liczbe rzeczywista: "))
        print(a)
    print(b)
```

Importowanie do listingu danych z pliku

Kod

```
\usepackage{color}
...
\definecolor{codegreen}{rgb}{0,0.6,0}
\definecolor{codegray}{rgb}{0.5,0.5,0.5}
\definecolor{codepurple}{rgb}{0.58,0,0.82}
\definecolor{backcolour}{rgb}{0.95,0.95,0.92}
...
```

Importowanie do listingu danych z pliku

Komenda `\lstdefinestyle{mystyle}{...}` definiuje własny styl dla listingu.

Kod

```
\lstdefinestyle{mystyle}{
    backgroundcolor=\color{backcolour},
    commentstyle=\color{codegreen},
    keywordstyle=\color{magenta},
    numberstyle=\tiny\color{codegray},
    stringstyle=\color{codepurple},
    basicstyle=\footnotesize,
    breakatwhitespace=false,
    breaklines=true,    captionpos=b,
    keepspaces=true,  numbers=left,
    numbersep=5pt,    showspaces=false,
    showstringspaces=false, showtabs=false,
    tabsize=2
}
```

Importowanie do listingu danych z pliku

Komenda `\lstset{style=mystyle}` umożliwia stosowanie zdefiniowanego przez siebie stylu do danego listingu.

Kod

```
\lstset{style=mystyle}
\lstinputlisting[language=Python,
firstline=3, lastline=9]{z2.py}
```

Importowanie do listingu danych z pliku

Wykonanie

```
1 def main():
2     n = int(input("Podaj ilosc liczb: "))
3     b = float(input("Podaj kolejna liczbe rzeczywista: ")
       )
4     for j in range(0, n-1):
5         a = float(input("Podaj kolejna liczbe rzeczywista:
        "))
6         print(a)
7     print(b)
```

Importowanie do listingu danych z pliku

Kod

```
\lstset{style=mystyle}
\begin{footnotesize}
\lstinputlisting[language=Python, firstline=3,
lastline=9,caption=Python example]{z2.py}
\end{footnotesize}
```

Importowanie do listingu danych z pliku

Wykonanie

```
1 def main():
2     n = int(input("Podaj ilosc liczb: "))
3     b = float(input("Podaj kolejna liczbe rzeczywista: ")
       )
4     for j in range(0, n-1):
5         a = float(input("Podaj kolejna liczbe rzeczywista:
        "))
6         print(a)
7     print(b)
```

Listing 1: Python example

Lista języków wspierana przez pakiet `listings`

Ada, Algol (60, 68), Ant, Assembler (Motorola68k, x86masm), Awk, bash, Basic, C (ANSI, Handel, Objective, Sharp), C++ (ANSI, GNU, ISO, Visual), Caml (light, Objective), CIL, Clean, Cobol (1974, 1985, ibm), Comal 80, command.com (WinXP), Comsol, csh, Delphi, Eiffel, Elan, erlang, Euphoria, Fortran (77, 90, 95), GCL, Gnuplot, Haskell, HTML, IDL (empty, CORBA), inform, Java (empty, AspectJ), JVMIS, ksh, Lingo, Lisp, Logo, make, Mathematica, Matlab, Mercury, MetaPost, Miranda, Mizar, ML, Modula-2, MuPAD, NASTRAN, Oberon-2, OCL (decorative, OMG), Octave, Oz, Pascal, Perl, PHP, PL/I, Plasm, PostScript, POV, Prolog, Promela, PSTricks, Python, R, Reduce, Rexx, RSL, Ruby, S (empty, PLUS), SAS, Scilab, sh, SHELXL, Simula (67, CII, DEC, IBM), SPARQL, SQL, tcl, TeX (AlLaTeX, common, LaTeX, plain, primitive), VBScript, Verilog, VHDL (empty, AMS), VRML, XML, XSLT.

Opcje, które można ustawić

backgroundcolor - kolor tła. Wymagane pakiety: `color` or `xcolor`.

commentstyle - styl dla komentarza.

basicstyle - styl czcionki rozmiar/rodzina (np. basicstyle=`\ttfamily\small`)

keywordstyle - styl dla słów kluczowych (np. keywordstyle=`\color{red}`)

numberstyle - styl dla numerów linii

numberserp - odległość numerów linii od kodu

stringstyle - styl dla łańcuchów

showspaces - wyróżnienie spacji w kodzie (true/false)

showstringspaces - wyróżnienie spacji w łańcuchach (true/false)

Opcje, które można ustawić - cd.

showtabs - wyróżnienie tabulatorów w kodzie (true/false)
numbers - pozycja numerów linii (left/right/none, np. no line numbers)
prebreak - wyświetlanie znaku końca linii (np. prebreak=↩)
captionpos - pozycja nagłówka (t/b)
frame - pokazywanie ramki na zewnątrz kodu (none/leftline/topline/bottomline/lines/single/shadowbox)
breakwhitespace - ustawiany, jeśli if automatyczne łamanie linii powinno wystąpić tylko w miejscu białego znaku.
breaklines - automatyczne łamanie linii.
keepspaces - zachowuje wcięcia w kodzie
tabsize - domyślny rozmiar tabulatora
escapeinside - określa znak ucieczki z kodu źródłowego do LaTeX-a (np. escapeinside=%**))
rulecolor - określa kolor ramki

Pakiet `algorithmic`

Pakiet `algorithmic` pozwala na formatowanie algorytmów za pomocą następujących poleceń:

```
\STATE <text>
\IF{<warunek>}\STATE{<tekst>}\ELSE\STATE{<tekst>}\ENDIF
\FOR{<warunek>}\STATE{<tekst>}\ENDFOR
\FOR{<warunek>\TO<warunek>}\STATE{<tekst>}\ENDFOR
\FORALL{<warunek>}\STATE{<tekst>}\ENDFOR
\WHILE{<warunek>}\STATE{<tekst>}\ENDWHILE
\REPEAT\STATE{<tekst>}\UNTIL{<condition>}
\LOOP \STATE{<tekst>}\ENDLOOP
\REQUIRE<tekst>
\ENSURE<tekst>
\RETURN<tekst>
\PRINT<tekst>,\COMMENT{<tekst>}
\AND, \OR, \XOR, \NOT, \TO, \TRUE, \FALSE
```

Pakiet `algorithmic`

Kod - Sortowanie przez proste wybieranie

```
\begin{algorithmic}[1]
\REQUIRE {tablica A o rozmiarze n}
\COMMENT{A=[0,\ldots,n-1]}
\FORALL{$i=0$ to $n-2$}
 \STATE $min = i$;
 \FORALL{$j=i+1$ to $n-1$}
    \IF {$A[j] < A[min]$}
      \STATE $min=j$;
    \ENDIF
    \STATE $j=j+1$;
 \ENDFOR
 \STATE {\bf zamiana (A, min, i)};
 \STATE $i=i+1;$
\ENDFOR
\end{algorithmic}
```

Pakiet `algorithmic`

Wykonanie - Sortowanie przez proste wybieranie

Require: tablica A o rozmiarze n {A=[0,. . . ,n-1]}

```
1:  for all i = 0 to n − 2 do
2:      min = i;
3:      for all j = i + 1 to n − 1 do
4:          if A[j] < A[min] then
5:              min = j;
6:          end if
7:          j = j + 1;
8:      end for
9:      zamiana (A, min, i);
10:     i = i + 1;
11: end for
```

Pakiet `algorithmic`

Kod - Sortowanie przez wstawianie

```
\begin{algorithmic}[1]
\REQUIRE {tablica A o rozmiarze n}
\COMMENT{A=[0,\ldots,n-1]}
\FOR{$i=1$ to $n-1$}
  \STATE $key = A[i]$;
  \STATE $j = i-1$;
  \WHILE{$j\geq 0$ and $A[j] > key$}
    \STATE $A[j+1] = A[j]$;
    \STATE $j=j-1$;
  \ENDWHILE
  \STATE $A[j+1] = key$;
\ENDFOR
\end{algorithmic}
```

Pakiet `algorithmic`

Wykonanie - Sortowanie przez wstawianie

Require: tablica A o rozmiarze n {A=[0,. . . ,n-1]}

1: **for** $i = 1$ to $n - 1$ **do**
2: $key = A[i]$;
3: $j = i - 1$;
4: **while** $j \geq 0$ and $A[j] > key$ **do**
5: $A[j + 1] = A[j]$;
6: $j = j - 1$;
7: **end while**
8: $A[j + 1] = key$;
9: **end for**

Pakiet `algorithm2e`

Podstawowe środowiska pakietu `algorithm2e`:

algorithm - składa algorytmy

algorithm* - składa algorytmy w trybie dwukolumnowym

procedure - składa procedury

procedure* - składa procedury w trybie dwukolumnowym

function - składa funkcje

function* - składa funkcje w trybie dwukolumnowym

Podstawowa opcja pakietu to **algo2e**:

```
\usepackage[algo2e]{algorithm2e}
```

Inne opcje:

noline - nie drukuje pionowej linii dla bloku.

lined - drukuje pionową linię dla bloku.

vlined - pionowa zakrzywiona linia dla bloku.

Pakiet `algorithm2e`

Kod

```
\begin{algorithm2e}[H]
\KwIn{Całkowite $a \geq 0$ i $b \geq 0$}
\KwOut{\textsc{NWD} z $a$ i $b$}
\While{$b \neq 0$}{
  $r \leftarrow a \bmod b$\;
  $a \leftarrow b$\;
  $b \leftarrow r$\;
}
\caption{Algorytm Euklidesa}
\end{algorithm2e}
```

Pakiet `algorithm2e`

Wykonanie

Input: Całkowite $a \geq 0$ i $b \geq 0$
Output: NWD z a i b
while $b \neq 0$ **do**
    $r \leftarrow a \bmod b$;
    $a \leftarrow b$;
    $b \leftarrow r$;
end

Algorithm 1: Algorytm Euklidesa

Pakiet `algorithm2e` - makra wejścia-wyjścia

Kod

```
\begin{algorithm2e}[H]
\KwIn{input}
\KwOut{output}
\KwData{input}
\KwResult{output}
\end{algorithm2e}
```

Wykonanie

Input: input
Output: output
Data: input
Result: output

Pakiet `algorithm2e` - makra słów kluczowych

Kod

```
\begin{algorithm2e}[H]
\KwTo
\KwRet{[value]}
\Return{[value]}
\Begin{block inside}
\end{algorithm2e}
```

Wykonanie

to
return *[value]*
return *[value]*
begin
  block inside
end

Pakiet `algorithm2e` - makra komentarzy w stylu C

Kod

```
\tcc{komentarz}
\tcc*{right justified side comment}
\tcc*[r]{komentarz wyrównany do prawej}
\tcc*[l]{komentarz wyrównany do lewej}
\tcc*[h]{komentarz wyrównany do lewej
bez nowej linii}
\tcc*[f]{komentarz wyrównany do prawej
bez nowej linii}
```

Pakiet `algorithm2e` - makra komentarzy w stylu C

Wykonanie

```
/* komentarz                                                    */
;              /* right justified side comment */
;            /* komentarz wyrównany do prawej */
;/* komentarz wyrównany do lewej                               */
/* komentarz wyrównany do lewej bez nowej
linii                                                           */
     /* komentarz wyrównany do prawej bez nowej
linii */
```

Pakiet `algorithm2e` - makra komentarzy w stylu C++

Kod

```
\tcp{komentarz}
\tcp*{right justified side comment}
\tcp*[r]{komentarz wyrównany do prawej}
\tcp*[l]{komentarz wyrównany do lewej}
\tcp*[h]{komentarz wyrównany do lewej
bez nowej linii}
\tcp*[f]{komentarz wyrównany do prawej
bez nowej linii}
```

Pakiet `algorithm2e` - makra komentarzy w stylu C++

Wykonanie

```
// komentarz
;                  // right justified side comment
;                 // komentarz wyrównany do prawej
;// komentarz wyrównany do lewej
// komentarz wyrównany do lewej bez nowej
linii
     // komentarz wyrównany do prawej bez nowej
linii
```

Pakiet `algorithm2e` - if

Kod

```
\begin{algorithm2e}[H]
\If{<warunek>}
{<ciało if-a>}
\end{algorithm2e}
```

Wykonanie

if *<warunek>* **then**
 | <ciało if-a>
end

Pakiet `algorithm2e` - if

Kod

```
\begin{algorithm2e}[H]
\uIf{<warunek>}
{<ciało if-a>}
\end{algorithm2e}
```

Wykonanie

if *<warunek>* **then**
| <ciało if-a>

Pakiet `algorithm2e` - if

Kod

```
\begin{algorithm2e}[H]
\lIf{<warunek>}
{<ciało if-a>}
\end{algorithm2e}
```

Wykonanie

if *<warunek>* **then** <ciało if-a>;

Pakiet `algorithm2e` - if

Kod

```
\begin{algorithm2e}[H]
\Else
{<ciało else>}
\end{algorithm2e}
```

Wykonanie

else
| <ciało else>
end

Pakiet `algorithm2e` - if

Kod

```
\begin{algorithm2e}[H]
\uElse
{<ciało else>}
\end{algorithm2e}
```

Wykonanie

else
| <ciało else>

Pakiet `algorithm2e` - if

Kod

```
\begin{algorithm2e}[H]
\lElse
{<ciało else>}
\end{algorithm2e}
```

Wykonanie

else <ciało else>;

Pakiet `algorithm2e` - if

Kod

```
\begin{algorithm2e}[H]
\ElseIf{<warunek>}
{<ciało else>}
\end{algorithm2e}
```

Wykonanie

else if *<warunek>* **then**
 | <ciało else>
end

Pakiet `algorithm2e` - if

Kod

```
\begin{algorithm2e}[H]
\uElseIf{<warunek>}
{<ciało else>}
\end{algorithm2e}
```

Wykonanie

else if *<warunek>* **then**
| <ciało else>

Pakiet `algorithm2e` - if

Kod

```
\begin{algorithm2e}[H]
\lElseIf{<warunek>}
{<ciało else>}
\end{algorithm2e}
```

Wykonanie

else if *<warunek>* **then** <ciało else>;

Pakiet `algorithm2e` - if

Kod

```
\begin{algorithm2e}[H]
\eIf{<warunek>}{<blok if>}{<blok else>}
\end{algorithm2e}
```

Wykonanie

if then
 | <blok if>
else
 | <blok else>
end

Pakiet `algorithm2e` - if

Kod

```
\begin{algorithm2e}[H]
\lIf{<warunek>}{<blok if>}{<blok else>}
\end{algorithm2e}
```

Wykonanie

if *<warunek>* **then** <blok if> ;
<blok else>

Pakiet `algorithm2e` - if

Kod

```
\begin{algorithm2e}[H]
\uIf{$a < 0$}{
\tcp{$a < 0$}
} \uElseIf{$a = 0$}{
\tcp{$a = 0$}
} \lElseIf{$a = 1$}{
\tcp{$a = 1$}
} {
\tcp{$a > 1$}
}
\end{algorithm2e}
```

Wykonanie

if $a < 0$ **then**
  // $a < 0$
else if $a = 0$ **then**
  // $a = 0$
else if $a = 1$ **then** // $a = 1$
;
// $a > 1$

Pakiet `algorithm2e`

Kod

```
\begin{algorithm2e}[H]
\Switch{<warunek>}
{<przypadki>}
\end{algorithm2e}
```

Wykonanie

switch *<warunek>* **do**
| <przypadki>
endsw

Pakiet `algorithm2e`

Kod

```
\begin{algorithm2e}[H]
\Case{<warunek>}
{<przypadki>}
\end{algorithm2e}
\begin{algorithm2e}[H]
\uCase{<warunek>}
{<przypadki>}
\end{algorithm2e}
```

Wykonanie

case *<warunek>*
| <przypadki>
end

case *<warunek>*
| <przypadki>

Pakiet `algorithm2e`

Kod

```
\begin{algorithm2e}[H]
\Case{<warunek>}
{<przypadki>}
\end{algorithm2e}
\begin{algorithm2e}[H]
\lCase{<warunek>}
{<przypadki>}
\end{algorithm2e}
```

Wykonanie

case *<warunek>*
| <przypadki>
end

case *<warunek>*
<przypadki>;

Pakiet `algorithm2e`

Kod

```
\begin{algorithm2e}[H]
\Other{<przypadki>}
\end{algorithm2e}
\begin{algorithm2e}[H]
\lOther{<przypadki>}
\end{algorithm2e}
```

Wykonanie

otherwise
| <przypadki>
end

otherwise <przypadki> ;

Pakiet `algorithm2e`

Kod

```
\begin{algorithm2e}[H]
\For{<warunek>}
{<ciało for>}
\end{algorithm2e}
\begin{algorithm2e}[H]
\lFor{<warunek>}
{<ciało for>}
\end{algorithm2e}
```

Wykonanie

for *<warunek>* **do**
| <ciało for>
end

for *<warunek>* **do** <ciało for>;

Pakiet `algorithm2e`

Kod

```
\begin{algorithm2e}[H]
\While{<warunek>}
{<ciało while>}
\end{algorithm2e}
\begin{algorithm2e}[H]
\lWhile{<warunek>}
{<ciało while>}
\end{algorithm2e}
```

Wykonanie

while *<warunek>* **do**
| <ciało while>
end

while *<warunek>* **do**
<ciało while>;

Pakiet `algorithm2e`

Kod

```
\begin{algorithm2e}[H]
\ForEach{<warunek>}
{<ciało for>}
\end{algorithm2e}
\begin{algorithm2e}[H]
\lForEach{<warunek>}
{<ciało for>}
\end{algorithm2e}
```

Wykonanie

foreach *<warunek>* **do**
| <ciało for>
end

foreach *<warunek>* **do**
<ciało for>;

Pakiet `algorithm2e`

Kod

```
\begin{algorithm2e}[H]
\ForAll{ <warunek> }
{<ciało for>}
\end{algorithm2e}
\begin{algorithm2e}[H]
\lForAll{ <warunek> }
{<ciało for>}
\end{algorithm2e}
```

Wykonanie

forall the *<warunek>* **do**
| <ciało for>
end

forall the *<warunek>* **do**
<ciało for>;

Pakiet `algorithm2e`

Kod

```
\begin{algorithm2e}[H]
\Repeat{<warunek końcowy>}
{<ciało repeat>}
\end{algorithm2e}
\begin{algorithm2e}[H]
\lRepeat{<warunek końcowy>}
{<ciało repeat>}
\end{algorithm2e}
```

Wykonanie

repeat
| <ciało repeat>
until *<warunek końcowy>*;

repeat <ciało repeat>
until *<warunek końcowy>*;

Pakiet `algorithm2e`

Więcej na temat pakietu `algorithm2e` można znaleźć na:
`ftp://ftp.gust.org.pl/TeX/macros/latex/contrib/algorithm2e/doc/algorithm2e.pdf`