



PAŃSTWOWA WYŻSZA SZKOŁA ZAWODOWA
W ELBLĄGU
INSTYTUT INFORMATYKI STOSOWANEJ

PRACA DYPLOMOWA INŻYNIERSKA

Preprocesor arytmetyczny
dla systemu składu tekstów $\text{\LaTeX}$

autor: **Kamil Piwowarczyk**

opiekun pracy: **dr Stefan Sokołowski**

Elbląg, 2009

Spis treści

Wstęp	2
1. System składu tekstów $\text{\LaTeX}$	3
1.1. $\text{\TeX}$ – fundament na którym opiera się $\text{\LaTeX}$	3
1.2. $\text{\LaTeX}$ – powstanie, zastosowanie i rozwój	4
1.3. Zasada działania $\text{\LaTeX}$ -a	6
2. Praca w $\text{\LaTeX}$-u	8
2.1. Procesor tekstu a system składu tekstu	8
2.2. Składnia $\text{\LaTeX}$ -a	8
2.2.1. Styl dokumentu	9
2.2.2. Pakiety	10
2.2.3. Podział treści dokumentu	11
2.3. Dlaczego warto używać $\text{\LaTeX}$ -a?	12
2.4. Porównanie $\text{\LaTeX}$ -a z innymi systemami składu drukarskiego	13
2.4.1. $\text{\ConTeXt}$	14
3. Arytmetyka w $\text{\LaTeX}$-u	15
3.1. Dostępne operacje na liczbach	15
3.1.1. Wymiary i odległości	15
3.1.2. Liczniki	16
3.2. Zastosowania wymagające obliczeń. Braki arytmetyczne systemu $\text{\LaTeX}$	16
3.3. Propozycja uzupełnienia systemu	19
4. Preprocesor arytmetyczny – $\text{\KTeX}$	20
4.1. Koncepcja wykonania	20
4.2. Realizacja systemu $\text{\KTeX}$	21
4.2.1. Gramatyka $\text{\KTeX}$ -a	23
4.2.2. Zmienne	25
4.2.3. Obliczenia arytmetyczne. Gramatyka wyrażeń języka $\text{\KTeX}$	26
4.2.4. Instrukcja warunkowa i operacje logiczne	28
4.2.5. Pętla <i>for</i>	30
4.2.6. Deklaracja i wywołanie funkcji	31
4.2.7. Obsługa błędów	33
4.3. Możliwości gotowego systemu	34
Podsumowanie	35
Spis rysunków	36
Spis tabel	37
Bibliografia	38
A. Gramatyka języka $\text{\KTeX}$	39
B. Dokument prezentacyjny – kod źródłowy	41
C. Dokument prezentacyjny – kod po przetworzeniu $\text{\KTeX}$-em	49
D. Dokument prezentacyjny – gotowy dokument	59
D.1. Przedstawienie składni systemu $\text{\KTeX}$	59
D.2. Praktyczne przykłady zastosowań $\text{\KTeX}$ -a	62
E. Zawartość CD	65

Wstęp

$\text{\LaTeX}$ jest systemem składu tekstów stanowiącym rozwinięcie systemu $\text{\TeX}$. System ten umożliwia automatyczne formatowanie raportów, artykułów czy książek. Z powodzeniem jest też używany do tworzenia prezentacji i grafiki wektorowej. Jest powszechnie wykorzystywany przy publikacjach naukowych, ponieważ umożliwia estetyczne składanie skomplikowanych wzorów i tabel oraz integrację rysunków z tekstem. Korzysta z niego bardzo wiele czasopism i wydawnictw naukowych. System jest już rozwijany od prawie 25 lat i powstało do niego wiele pakietów dodatkowych, umożliwiających skład między innymi: muzyki – nut, poezji – z użyciem czcionek pisma odręcznego, z różnymi wariantami liter ozdobnych; plakatów, prezentacji interaktywnych itd.

W pracy tej przedstawiona została krótka historia $\text{\LaTeX}$ -a, idea pracy z tym systemem oraz porównanie z innymi systemami składu drukarskiego. Zaprezentowano także zastosowania, których nie da się w pełni wykorzystać z powodu braków arytmetycznych $\text{\LaTeX}$ -a – niemożliwości obliczania wyrażeń ani implementacji algorytmu.

Głównym celem pracy było stworzenie rozszerzenia do $\text{\LaTeX}$ -a, umożliwiającego wykonywanie wspomnianych obliczeń. W efekcie powstał autorski system – preprocesor arytmetyczny, nazwany od imienia autora – $\text{\KTeX}$. System, na który składa się język wraz z kompilatorem, umożliwia zaprogramowanie obliczeń – w pliku źródłowym $\text{\LaTeX}$ -a – i wykorzystanie ich wyników w treści, bądź składzie dokumentu. Koncepcji wykonania i realizacji preprocesora $\text{\KTeX}$ poświęcono rozdział 4 pracy, natomiast przykłady użycia przedstawiono w załącznikach B–D.

1. System składu tekstów $\text{\LaTeX}$

1.1. $\text{\TeX}$ – fundament na którym opiera się $\text{\LaTeX}$

Omówienie systemu $\text{\LaTeX}$ (wymowa *latech* lub *lejtech*) wymaga przedstawienia systemu, który stanowi jego podstawę, czyli systemu profesjonalnego składu drukarskiego $\text{\TeX}$. Nazwa $\text{\TeX}$ wzięła się od liter τ (*tau*), ϵ (*epsilon*) oraz χ (*chi*) – pierwszych trzech liter greckiego słowa *technē* (gr. $\tau\epsilon\chi\nu\eta$, od niego pochodzi wyraz *technika*), oznaczającego sztukę lub rzemiosło. Z pochodzenia nazwy wynika poprawna wymowa: *tech*, a nie *tex*, która jest jednak dość trudna dla ludzi posługujących się językiem angielskim – dlatego przeważa u nich wymowa *tek*.

System $\text{\TeX}$ został stworzony przez Amerykanina Donalda E. Knutha. Profesor Knuth, informatyk i matematyk, nie był zadowolony z jakości wydania swojej książki „Sztuka Programowania”¹⁾. Nie znajdując odpowiednich narzędzi postanowił sam stworzyć oprogramowanie do formatowania tekstu, które będzie zarówno funkcjonalne, jak i spełni wszelkie wymogi estetyczne. Mimo założeń, że system powstanie w pół roku, Knuthowi osiem lat zajęło rozwijanie programu do finalnej wersji. Od tamtej pory, czyli od roku 1985, poprawiane są tylko błędy.

Nazwa $\text{\TeX}$ odnosi się do dedykowanego języka jak i kompilatora generującego plik nadający się do wyświetlenia lub wydruku. System w wersji od *TeX82*²⁾ pisany był w języku *WEB*, również autorstwa Knutha. Język *WEB* opiera się założeniu, że programy komputerowe powinny być zrozumiałe dla ludzi. Knuth określił tę koncepcję jako *literate programming*³⁾. W przeciwieństwie do powszechnego trendu w programowaniu,

¹⁾ Tytuł oryginalny: „The Art of Computer Programming”; w Polsce wydana w czterech tomach przez Wydawnictwa Naukowo-Techniczne.

²⁾ Nazwa *TeX82* pochodzi od roku, w którym wydano tą wersję – 1982. W późniejszym czasie Knuth przyjął inny sposób numeracji wersji $\text{\TeX}$ -a – w 1989 r. ukazała się wersja 3.0, natomiast kolejne oznaczane są coraz dokładniejszym przybliżeniem liczby π (aktualna wersja, datowana marzec 2008, ma numer: 3.1415926).

³⁾ ang. *literate* – w dosłownym tłumaczeniu: człowiek piśmienny, ale w znaczeniu nie tyle umiejącego pisać i czytać, co wykształconego (w przeciwieństwie do człowieka niepiśmiennego).

gdzie najważniejszy jest działający kod, a dokumentacja stanowi dodatek, tutaj najważniejszy jest opis, w który wplecione są fragmenty kodu w języku programowania. Język ten powstał właśnie po to, aby zaprezentować kod źródłowy $\text{\TeX}$ -a. *WEB* generuje programy w języku *Pascal* i dokumentację w języku $\text{\TeX}$, z czasem powstała wersja *CWEB* – generująca programy w *C*.

Częścią $\text{\TeX}$ -a jest, używany do dzisiaj, program do opisu czcionek – **METAFONT**, również autorstwa Knutha i również napisany w języku *WEB*⁴⁾.

$\text{\TeX}$ ma zastosowanie do publikacji naukowych – matematycznych, fizycznych, informatycznych itd. – ze względu na możliwość tworzenia rozbudowanych tabel i skomplikowanych wzorów. Jest dostępny na praktycznie wszystkie platformy sprzętowe i systemowe, pozwala to uzyskać ten sam efekt przy publikacji na dowolnym komputerze. W przedmowie do książki [3], Gordon Bell napisał ponad dwadzieścia lat temu:

Tau Epsilon Chi ($\text{\TeX}$) Dona Knutha jest potencjalnie najbardziej znaczącą rzeczą w dziedzinie składu tekstu, w tym wieku. Wprowadza on [$\text{\TeX}$] standard w komputerowej typografii i pod względem wagi może równać się z wprowadzeniem prasy Gutenberga.

Słowa te zdaje się potwierdzać fakt, że $\text{\TeX}$ stał się podstawowym standardem w przetwarzaniu tekstu w wielu uczelniach i laboratoriach badawczych, a także jest używany przez profesjonalne wydawnictwa.

Dzięki temu, że $\text{\TeX}$ jest rozpowszechniany nieodpłatnie, z dostępem do plików źródłowych (które, jak już zostało wspomniane, są połączone z czytelną dokumentacją), powstało na jego bazie wiele rozszerzeń i systemów. Jednym z najpopularniejszych z nich jest właśnie $\text{\LaTeX}$.

1.2. $\text{\LaTeX}$ – powstanie, zastosowanie i rozwój

$\text{\LaTeX}$ jest systemem opracowywania dokumentów bazującym na systemie $\text{\TeX}$. Poprawny zapis nazwy musi być sformatowany w formie loga – tak jak w całej tej

⁴⁾ Pierwsza wersja **METAFONT**-a, która powstała w 1977 r. nie była jednak stworzona w *WEB* – program został przepisany w tym języku dopiero w 1984 r. i od tego roku jego wersje numerowane są kolejnymi przybliżeniami liczby Eulera (aktualna wersja, datowana na marzec 2008, to: 2.718281).

pracy, jednak wszędzie gdzie nie da się go zastosować przyjmuje się zapis LaTeX. System ten stanowi zestaw makr i dodatkowych pakietów rozbudowujących T_EX-a, ale przede wszystkim automatyzujących proces składania tekstu.

W porównaniu składu np. artykułu, przy użyciu L^AT_EX-a, z tradycyjnym procesem przygotowania tekstu do publikacji, L^AT_EX stanowi odpowiednik projektanta graficznego. Projektant graficzny decyduje, w zależności od przeznaczenia tekstu, o wyglądzie dokumentu – wybiera kroje czcionek, odstęp między tytułami, szerokość tekstu na stronie (ewentualnie w kolumnie, jak i samą ilość kolumn) itp. Sporządza następnie szczegółowe instrukcje dla zecera⁵⁾, który dawniej przygotowywał matryce drukarskie, a teraz, w dobie informatyzacji, pliki komputerowe. Obecnie funkcja składacza może być połączona z funkcją projektanta graficznego – jedna osoba przygotowuje cały projekt graficzny gotowy do druku. W pracy w L^AT_EX-u zecerem jest T_EX, któremu L^AT_EX przekazuje jak dokument ma zostać złożony.

Autorem L^AT_EX-a jest Leslie Lamport i to od pierwszych dwóch liter jego nazwiska pochodzi nazwa systemu. Lamport napisał pierwszą wersję systemu na początku lat osiemdziesiątych. Jego dzieło szybko stało się podstawowym sposobem korzystania z systemu T_EX. Z powodu popularności systemu Lamporta stosuje się do jego określania zamiennie samą nazwę T_EX. W [4, str. 2] – książce-przewodniku po L^AT_EX-u – czytamy:

Obecnie L^AT_EX jest nadzwyczaj popularny w środowisku naukowym i akademickim, a także powszechnie stosowany w działalności wydawniczej. Stał się *lingua franca* świata nauki; w postaci dokumentów latechowych naukowcy wysyłają swoje prace drogą elektroniczną kolegom na całym świecie.

Czasopisma naukowe, przyjmujące artykuły w L^AT_EX-u, udostępniają swoje szablony, dzięki którym autor wie jak jego tekst będzie się prezentował w piśmie. Bardzo mało osób pracuje w „czystym” T_EX-u, który w porównaniu do L^AT_EX-a jest językiem niskiego poziomu. Jednak bez znajomości podstawy na jakiej opiera się L^AT_EX nie można tworzyć nowych wzorów dokumentów, czy zastosować niestandardowego formatowania.

⁵⁾ zecer, składacz (od *niem. setzen* – składać, usadzać) – wysoko kwalifikowany pracownik zecerni wykonujący skład ręczny lub maszynowy na potrzeby druku typograficznego.

Pierwsza, publicznie udostępniona w roku 1985, wersja L^AT_EX-a opatrzona była numerem 2.09. Dynamiczny rozwój tego systemu spowodował pojawianie się wielkiej liczby nieoficjalnych rozszerzeń. Część z nich łamała ograniczenia systemu i nie we wszystkich instalacjach otrzymywano identyczny efekt ich działania. Koniecznością stało się wprowadzenie nowej wersji L^AT_EX-a, która miała usystematyzować rozszerzenia i wprowadzić nową funkcjonalność. W 1994 r. oddano do użytku wersję L^AT_EX 2_ε, w której wprowadzono nowe polecenia (w tym pozwalające na zmianę kolorów czy umieszczanie w tekście grafiki), a także poprawiono wiele błędów i wprowadzono dużo ulepszeń. Od lat '90 trwają prace nad kolejną wersją – L^AT_EX3, która według oficjalnego dokumentu [6] ma oferować udoskonaloną składnię, wsparcie dla hiperłączy, dostęp do zewnętrznych czcionek oraz nowy interfejs użytkownika. Część z tych elementów została już udostępniona w postaci pakietów do wersji L^AT_EX 2_ε.

1.3. Zasada działania L^AT_EX-a

Rozpoczęcie pracy w L^AT_EX-u musi zostać poprzedzone instalacją pakietu zawierającego systemy T_EX, METAFONT, samego L^AT_EX-a oraz inne narzędzia i pakiety. W przypadku *Linux*-a, L^AT_EX jest dostępny w repozytoriach, natomiast dla systemów operacyjnych *Microsoft Windows* jedną z najpopularniejszych dystrybucji jest MiK_TE_X⁶⁾. Innym pakietem jest T_EXLive – wydanie na płycie DVD zawierające wersje instalacyjne dla *Linux*-a, *Mac OS* i *Windows*⁷⁾.

Aby stworzyć dokument L^AT_EX-a należy najpierw przygotować plik źródłowy z rozszerzeniem `.tex`⁸⁾, posługując się w tym celu dowolnym edytorem tekstu (przykład pliku źródłowego – wraz z omówieniem składni i zasad jego pisania – został przedstawiony w 2.2). Następnie taki plik poddaje się kompilacji poleceniem systemowym `latex`, w wyniku czego tworzone są dodatkowe pliki pomocnicze i plik wynikowy DVI. Wykaz plików, wymaganych i tworzonych przez T_EX-a i L^AT_EX-a, oraz ich funkcje został przedstawiony w tabeli 1.1. Znajomość tych plików i ich funkcji nie jest wymagana

⁶⁾ Strona projektu: <http://miktex.org>

⁷⁾ T_EXLive na stronach T_EX User Group (TUG): <http://www.tug.org/texlive/>, i polskiej Grupy Użytkowników Systemu T_EX (GUST): http://www.gust.org.pl/doc/tex_install/

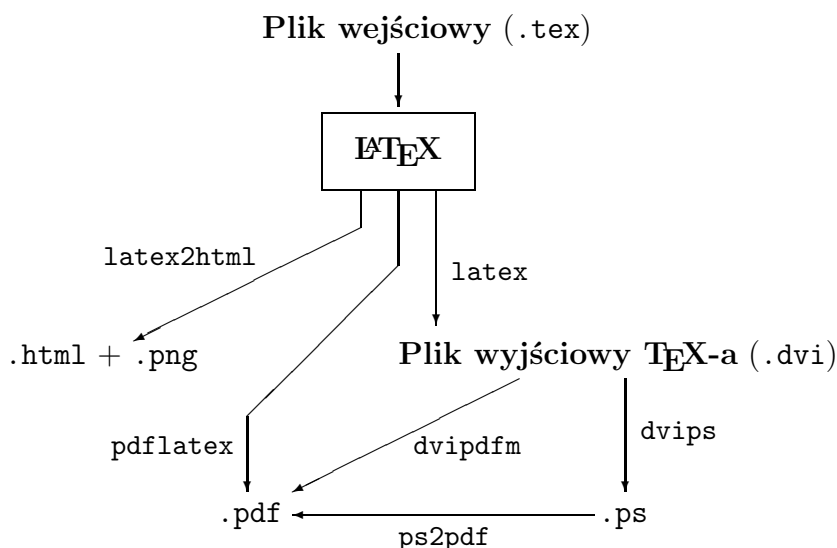
⁸⁾ Jest to domyślne rozszerzenie pliku wejściowego przy kompilacji (używane w przypadku podania samej nazwy, bez rozszerzenia), jednak nie jest ono obowiązkowe – można używać dowolnego rozszerzenia, jednak trzeba je wtedy podawać przy kompilacji.

do pracy w $\text{\LaTeX}$ -u, warto jednak zwrócić uwagę na pliki `.aux`, `.log`, czy np. `.toc`, ponieważ tworzone są przy kompilacji w katalogu z plikiem źródłowym.

Tabela 1.1. Pliki używane przez $\text{\LaTeX}$ -a
Źródło: opracowanie własne na podstawie [1, str. 4]

Funkcja	Rozszerzenie
Plik źródłowy $\text{\LaTeX}$ -a	<code>.tex</code> , <code>.ltx</code>
Plik wynikowy sformatowany $\text{\TeX}$ -em	<code>.dvi</code>
Pliki dzienników	<code>.log</code> , <code>.texlog</code> , <code>.lis</code> , <code>.list</code>
Plik źródłowy METAFONT	<code>.mf</code>
Pliki związane z czcionkami	<code>.fd</code> , <code>.pk</code> , <code>.tfm</code>
Plik puli łańcucha (<i>string</i>)	<code>.pool</code> , <code>.poo</code> , <code>.pol</code>
Plik formatu	<code>.fmt</code>
Plik układu – klasy i pakiety $\text{\LaTeX}$ -a	<code>.clo</code> , <code>.cls</code> , <code>.dtx</code> , <code>.sty</code>
Plik pomocniczy $\text{\LaTeX}$ -a	<code>.aux</code>
Pliki spisu treści, ilustracji i tabel	<code>.toc</code> , <code>.lof</code> , <code>.lot</code>
Pliki skojarzone z $\text{\BibTeX}$ -em	<code>.bbl</code> , <code>.bib</code> , <code>.blg</code> , <code>.bst</code>
Pliki indeksu	<code>.idx</code> , <code>.ilg</code> , <code>.ind</code> , <code>.ist</code>

Rozszerzenie `.dvi` pliku wynikowego pochodzi od jego nazwy – plik *niezależny od urządzenia wyjściowego* (ang. *device-independent*). Plik ten można obejrzeć programem do podglądu (dołączonym do dystrybucji $\text{\TeX}$ -a), wykorzystać do wydruku na drukarce, czy przetworzyć na inny format, taki jak *PostScript* (`.ps`), *Portable Document Format* (`.pdf`), czy *HTML*. Rysunek 1.1 ilustruje możliwe sposoby uzyskania wymienionych formatów plików, z uwzględnieniem poleceń jakie można wykorzystać do ich utworzenia.



Rysunek 1.1. Polecenia kompilacji w $\text{\LaTeX}$ -u
Źródło: opracowanie własne na podstawie [8, str. 1]

2. Praca w L^AT_EX-u

2.1. Procesor tekstu a system składu tekstu

L^AT_EX nie jest procesorem tekstu tak jak *Microsoft Word*, czy *OpenOffice Writer*. Procesory tekstu opierają się na zasadzie **WYSIWYG** (ang. *What You See Is What You Get*), gdzie to, co widzimy na ekranie w trakcie pisania, będzie wyglądało tak samo, lub bardzo podobnie, na wydruku. Dominuje więc tutaj projektowanie wizualne: jeżeli autor chce wyróżnić tytuł, to musi sam ustawić odpowiednią czcionkę (np. *Times New Roman* w kroju pogrubiona kursywa, 24 pt) i interlinia (np. 1,5 linii odstępu między wierszami, plus dodatkowo 10 pt przed pierwszym akapitem).

L^AT_EX natomiast opiera się na idei **WYSIWYM** (ang. *What You See Is What You Mean*): tutaj autor musi ustalić jedynie logiczną strukturę treści, a jej formatowaniem zajmie się L^AT_EX. Idea ta wynika z założenia, że autor powinien skupić się na jak najlepszym i najbardziej zrozumiałym przekazaniu treści odbiorcy. Aby tego dokonać, nie powinien rozpraszać się wyglądem sporządzanego dokumentu, a zająć się sensem i logicznym układem tekstu. Leslie Lamport w [4, str. 8] określił podejście do składu tekstów w L^AT_EX-u jako właśnie *projektowanie logiczne* (ang. *logical design*).

2.2. Składnia L^AT_EX-a

Jak wspomniano w 1.2, L^AT_EX spełnia funkcję projektanta graficznego decydującego o wyglądzie dokumentu. Człowiek, wykonujący tę pracę, wie co jest tematem tekstu i jest w stanie stwierdzić co jest tytułem, co cytatem, a co np. kodem źródłowym programu komputerowego, i nadać im odpowiednie formatowanie. L^AT_EX-owi wszystkie te rzeczy trzeba wskazać, za pomocą *komend* i *środowisk*. Komendy L^AT_EX-a mają postać:

```
\nazwa_polecenia[opcje]{parametry}
```

Natomiast środowiskiem nazywa się obszar, o specyficznym formatowaniu, utworzony komendami:

```
\begin{nazwa_środowiska}
...
\end{nazwa_środowiska}
```

Dokument L^AT_EX-a składa się z preambuły i następującej po niej treści samego dokumentu – zamkniętej w środowisku *document*. Schemat prostego dokumentu wygląda następująco:

```
\documentclass[opcje]{klasa_stylu}

\usepackage{nazwa_pakietu}
...
\title{Tytuł}
\author{Jan Kowalski}
\date{\today}

\begin{document}
    treść dokumentu
    ...
\end{document}
```

Preambułą, czyli nagłówkiem jest cały początek pliku, aż do wystąpienia środowiska *document*, zawiera on definicję stylu dokumentu, komendy dołączające dodatkowe pakiety oraz informacje o autorze, dacie itp. W następnych podrozdziałach omówione zostaną podstawowe komendy, używane w preambule oraz komendy odpowiedzialne za podział treści dokumentu.

2.2.1. Styl dokumentu

Na wstępie pliku źródłowego, w preambule, znajduje się komenda definiująca rodzaj stylu dokumentu, czyli jego klasę. Komenda ta ma postać:

```
\documentclass[opcje]{klasa_stylu}
```

Najczęściej stosowane klasy dokumentów to **article** – artykuł, **book** – książka, **report** – raport, **slides** – slajdy/prezentacja, **letter** – list. Istnieją też specjalne – spolszczone klasy dokumentów, stworzone przez Marcina Wolińskiego, wchodzące w skład pakietu P_EL_AT_EX, autorstwa wymienionego M. Wolińskiego oraz Mariusza Olko. Dostępne klasy to **mwart** – artykuł, **mwbk** – książka, **mwrep** – raport; wprowadzają one polskie zwyczaje typograficzne, takie jak wcięcie pierwszego akapitu rozdziału/sekcji, brak dzielenia

wyrazów w tytułach, czy wygląd przypisów. Więcej o tym pakiecie można przeczytać np. w [5, str.5].

W komendzie `\documentclass`, w polu opcji, można ustalić też:

- format papieru (w normie ISO 216 np. `a4paper`, `a5paper`, `b5paper` itd., a w standardzie U.S. np. `letterpaper`, `legalpaper`, `executivepaper`),
- podstawowy rozmiar czcionki (np. `10pt` lub `12pt` – rozmiar ten można zwiększać i zmniejszać w treści pracy poleceniami np. `\large`, `\small`, `\scriptsize`),
- odmienne formatowanie stron parzystych i nieparzystych (opcja `oneside` odpowiada takiemu samemu formatowaniu wszystkich stron, natomiast `twoside` – różnemu; jest to wykorzystywane do przygotowywania druku dwustronnego, gdzie wymagane jest pozostawienie większego marginesu wewnętrznego na zszycie, czy np. różne wyrównanie numeracji lub inna zawartość paginy),
- sposób wyrównania wzorów i położenie ich numeracji (`fleqn` – wzory wyrównane do lewej zamiast do środka strony; `leqno` – numeracja przy lewym marginesie zamiast przy prawym),
- osobną stronę tytułową (`titlepage`, `notitlepage`),
- przeznaczenie wyniku kompilacji (`final` – daje ostateczny skład, `draft` – daje podgląd – szkic, który ułatwia wykrycie błędów w składzie).

Poniżej przedstawiono przykład preambuły jaka została użyta do stworzenia tego dokumentu:

```
\documentclass[a4paper,12pt,oneside]{mwrep}
```

Użyto tutaj wspomnianej wcześniej klasy Marcina Wolińskiego – `mwrep`. Zastosowano styl raportu ponieważ odpowiada on składowi pracy dyplomowej, obowiązującemu w *PWSZ w Elblągu* pod względem stylu numeracji sekcji, formatowania ich tytułów i podziału stron.

2.2.2. Pakiety

Po zdefiniowaniu ogólnego stylu dokumentu można dołączyć dodatkowe pakiety poleceniem:

```
\usepackage[opcje]{nazwa_pakietu}
```

Jest to konieczne przy pisaniu w języku innym niż angielski, czy przy dołączaniu do tekstu rysunków. Najczęściej stosowane pakiety:

- **polski** – pakiet języka polskiego (wchodzący w skład wspomnianego $\text{P}\text{L}\text{A}\text{T}\text{E}\text{X}$ -a), zalecany w miejsce standardowego, wielojęzycznego pakietu **babel**, który cechuje gorsza jakość polskich znaków.
- **graphicx** – pakiet umożliwiający wstawianie grafiki. Polecanym dla $\text{L}\text{A}\text{T}\text{E}\text{X}$ -a formatem graficznym jest *eps* (czyli z ang. *Encapsulated Postscript*). Format ten umożliwia przechowywanie zarówno grafiki wektorowej jak i rastrowej oraz jest praktycznie niezależny od sprzętu czy oprogramowania. Możliwe jest także wstawianie grafiki w innych formatach jak *jpeg*, czy *png*.
- **inputenc** – pakiet do wyboru kodowania znaków. Jako opcję podaje się nazwę tablicy kodowej, dostępne kodowania to np. *latin2* dla systemów *Linux/Unix*, *cp1250* dla *Windows* lub między-platformowe *utf8*.
- **amsmath** – moduł matematyczny, pozwalający na zaawansowane formatowanie wzorów, układów równań itp.¹⁾.
- **amssymb** – symbole matematyczne.

W nagłówku można ustalić też informacje o tytule i autorze pracy. Wprowadza się je komendami `\title` i `\author`, opcjonalnie można wprowadzić także datę poleceniem `\date`. Z tych informacji można wygenerować stronę tytułową lub nagłówek pracy, w zależności od wybranej klasy dokumentu i opcji `\titlepage` lub `\notitlepage`. Wykonuje się to poleceniem `\maketitle`, które tak jak i cała treść pracy, musi się znaleźć się w środowisku *document*.

2.2.3. Podział treści dokumentu

Treścią dokumentu będzie wszystko co znajduje się w środowisku *document*. Na początku oprócz wymienionego `\maketitle`, można użyć też polecenia generującego spis treści – `\tableofcontents`. Pozycje spisu treści tworzone są automatycznie z tytułów rozdziałów, na które podzielono pracę. Podziału treści można dokonać za pomocą następujących komend:

¹⁾ Strona internetowa projektu: <http://www.ams.org/tex/amslatex.html>

- `\part` – podział na części z osobną numeracją stron i rozdziałów (tylko dla klas `book` i `report`),
- `\chapter` – rozdział (tylko dla klas `book` i `report`),
- `\section`, `\subsection`, `\subsubsection` – podrozdziały,
- `\paragraph`, `\subparagraph` – akapity,
- `\appendix` – załącznik – rozpoczyna osobną numerację rozdziałów i sekcji.

Wszystkie komendy, poza załącznikiem, posiadają dwa argumenty: w nawiasach sześciennych podaje się tytuł, a w nawiasach kwadratowych opcjonalnie tytuł pomocniczy, który będzie widoczny w miejsce właściwego tytułu w spisie treści i na paginie. Podstawy tworzenia dokumentów w $\text{\LaTeX}$ -u opisane są w przetłumaczonym na język polski dokumencie [7].

2.3. Dlaczego warto używać $\text{\LaTeX}$ -a?

Główną zaletą stosowania $\text{\LaTeX}$ -a jest wysoka jakość składu drukarskiego, w tym celu, jak wspomniano w 1.1, powstał system $\text{\TeX}$. Jednak pisanie w $\text{\LaTeX}$ -u wymaga nauczania się składni języka i wielu komend, więc wydaje się, że nie jest to ani efektywne, ani opłacalne. Procesory tekstu nadają się do sporządzenia krótkich dokumentów, ale przy rozbudowanych pracach np. książkach, utrzymanie jednolitego formatowania staje się bardzo kłopotliwe i zabiera zdecydowanie za dużo czasu. Kiedy pozna się możliwości $\text{\LaTeX}$ -a, pracuje się w nim o wiele efektywniej niż w procesorach tekstu. W $\text{\LaTeX}$ -u w sposób automatyczny tworzone są np.:

- spisy treści, ilustracji oraz tabel,
- numerowanie i referencje do rozdziałów, podrozdziałów oraz elementów takich jak wzory, tabele i rysunki,
- skorowidze, indeksy,
- bibliografie i odwołania do literatury.

Główna zaleta procesorów tekstu, czyli graficzny interfejs i podgląd końcowego efektu pracy, została też po części przełamana wraz z pojawieniem się edytorów do $\text{\LaTeX}$ -a.

Najbardziej rozpowszechnione z nich to *LEd* (*LaTeX Editor*) i *TeXnicCenter* pod system operacyjny *Windows* oraz *Kile* i *LyX* pod systemy *Linux*. Zasadą działania przypominają one edytory *HTML* tzn. posiadają opcje podpowiedzi i autouzupełniania składni oraz podgląd gotowego dokumentu. *LyX* pozwala na pracę w trybie *WYSIWYG*, gdzie $\text{\LaTeX}$ zajmuje się składem pracując w tle, w trakcie pisania.

2.4. Porównanie $\text{\LaTeX}$ -a z innymi systemami składu drukarskiego

Programy przeznaczone do składu drukarskiego różnią się możliwościami i poziomem skomplikowania od standardowych procesorów tekstu, jednak większość komercyjnych aplikacji tego typu także opiera się na interfejsie graficznym i zasadzie *WYSIWYG*. Najbardziej znanymi programami składu tekstu są obecnie konkurencyjne *Adobe InDesign* i *QuarkXPress*.

QuarkXPress istnieje na rynku od 1987 roku. Pozwala na skomplikowany skład z dużą ilością grafiki z zastosowaniem do czasopism i „kolorowych” książek. W programie tym można tworzyć też strony internetowe, a nawet całe serwisy w kodzie *HTML* z zastosowaniem *CSS*. Możliwe jest także tworzenie animacji i stron w standardzie *Flash*.

InDesign początkowo stanowił kontynuację popularnego programu do składu tekstu *PageMaker*. Obecnie program wchodzi w skład *Adobe Creative Suite (CS)*, pakietu zawierającego takie aplikacje jak *Photoshop*, *Flash* czy *Dreamweaver*, jest więc przystosowany do bliskiej z nimi współpracy. Podstawowy pakiet *CS* ma możliwości takie jak *QuarkXPress*, na konkurowanie z którym firma *Adobe* była od początku nastawiona.

Oba programy klasyfikowane są jako systemy składu i łamania tekstu, jednak opierają się na innej formie pracy niż $\text{\LaTeX}$, a przede wszystkim są przeznaczone do innych zadań. Możliwe jest w nich składanie kolorowych miesięczników i zwykłych książek, ale nie prac naukowych, zawierających wzory matematyczne. Przeznaczone są dla profesjonalnych grafików i projektantów, i można powiedzieć, że stanowią narzędzie „ręcznego” składu tekstu.

2.4.1. ConT_EXt

ConT_EXt jest systemem przygotowywania dokumentów, który tak jak L^AT_EX bazuje na systemie T_EX. Został stworzony przez Holendra Hansa Hageny około 1991 r., w celu zautomatyzowania i uproszczenia procesu przygotowywania tekstu z pomocą T_EX-a, czyli z tych samych powodów co L^AT_EX.

System ten posiada podobną składnię jeśli chodzi o podziały sekcji, ma też takie same możliwości tworzenia spisów treści, bibliografii oraz referencji i odwołań. Różnice w założeniach systemu są jednak widoczne już przy tworzeniu prostego dokumentu. Podczas gdy w L^AT_EX-u autor decyduje o logicznym podziale pracy i wybiera jedynie szablon stylu dokumentu, w systemie ConT_EXt nie jest wymagane korzystanie z szablonu, trzeba jednak samemu zdefiniować wygląd wszystkich nagłówek, tytułów itp. ConT_EXt oferuje pełną kontrolę nad wyglądem dokumentu w miejsce z góry zdefiniowanych estetycznych standardów, które dostaje użytkownik L^AT_EX-a. W L^AT_EX-u można zmienić definicję wszystkich makr i w efekcie wpłynąć na wygląd dokumentu, jednak może to czasem wymagać sporo pracy. Często też wymagana jest znajomości składni T_EX-a, jednak ułatwienie stanowi tu bardzo dobra dokumentacja kodu źródłowego.

Język ConT_EXt od początku powstawał z myślą o tworzeniu graficznych, interaktywnych prezentacji. Przykładem takiej prezentacji może być [2] – instrukcja do tego systemu. L^AT_EX od wersji L^AT_EX 2_ε zyskał pakiety, które także pozwalają na operowanie kolorami i tworzenie interaktywnych odsyłaczy wewnątrz dokumentu (hiperłączy do rozdziałów, rysunków, wzorów czy pozycji bibliografii); możliwe jest też tworzenie prezentacji na równie wysokim poziomie dzięki pakietowi *Beamer*.

Jako główną zaletę ConT_EXt-a wymienia się też prostą, bardzo intuicyjną składnię, jednak jeśli chodzi o dokumentację kodu źródłowego, a co za tym idzie możliwość zmian i rozbudowy systemu u podstaw, to L^AT_EX jest uznawany za lepszy wybór.

3. Arytmetyka w $\text{\LaTeX}$ -u

3.1. Dostępne operacje na liczbach

$\text{\LaTeX}$ pozwala łatwo łączyć fragmenty kodu źródłowego, algorytmów i wzorów matematycznych w spójny estetycznie dokument tekstowy. Aby tego dokonać przy kompilacji pliku źródłowego, $\text{\TeX}$ dokonuje wielu przeliczeń odległości i położenia elementów takich jak tytuły, całe wyrazy czy pojedyncze litery. Dokonywane jest też liczenie kolejnych rozdziałów, rysunków i wzorów.

3.1.1. Wymiary i odległości

W $\text{\LaTeX}$ u można wpływać między innymi na następujące wielkości:

- szerokość ($\text{\texttt{\textbackslash textwidth}}$) i wysokość ($\text{\texttt{\textbackslash textheight}}$) tekstu na stronie,
- marginesy: górny ($\text{\texttt{\textbackslash topmargin}}$), lewy margines na stronach nieparzystych ($\text{\texttt{\textbackslash oddsidemargin}}$) i parzystych ($\text{\texttt{\textbackslash evensidemargin}}$)¹⁾,
- wymiary paginy górnej ($\text{\texttt{\textbackslash headheight}}$, $\text{\texttt{\textbackslash headsep}}$) i dolnej ($\text{\texttt{\textbackslash footskip}}$).

Polecenia podane w nawiasach służą do pobrania wartości danego parametru. Zmiany tych wartości można dokonać na kilka sposobów np.:

- $\text{\texttt{\textbackslash textwidth}}$ wartość – polecenie $\text{\TeX}$ -a, przypisuje wymiarowi podaną wartość,
- $\text{\texttt{\textbackslash setlength\{\textbackslash textwidth\}\{wartość\}}}$ – komenda $\text{\LaTeX}$ -a ustalająca wartość,
- $\text{\texttt{\textbackslash addtolength\{\textbackslash textwidth\}\{wartość\}}}$ – komenda $\text{\LaTeX}$ -a dodaje do wybranej wielkości podaną wartość (dopuszczalne są wartości ujemne).

Wszystkie wielkości mogą być zdefiniowane z użyciem jednostek metrycznych (np. *mm*, *cm*), US (np. *in*) lub drukarskich (np. *pt*, *em*).

¹⁾ Lewy margines wraz z ustaleniem szerokości tekstu pozwala dowolnie zdefiniować oba marginesy strony.

3.1.2. Liczniki

Polecenia podziału treści, które zostały wymienione w 2.2.3, przy każdym wywołaniu zwiększają skojarzony z nimi licznik numeracji. Przy interpretacji polecenia np. utworzenia nowego rozdziału – `\chapter{Tytuł}`, $\text{\LaTeX}$ najpierw zwiększy licznik o nazwie *chapter*, a następnie użyje go do wygenerowania nagłówka rozdziału. Inne liczniki to np.:

- *page* – strona,
- *figure* – rysunek,
- *table* – tabela,
- *equation* – równanie matematyczne,
- *footnote* – przypis,
- *enumi*, *enumii*, *enumiii* i *enumiv* – wyliczenia numerowane (*i*, *ii*, *iii*, *iv* oznaczają kolejne zagłębienia wyliczeń).

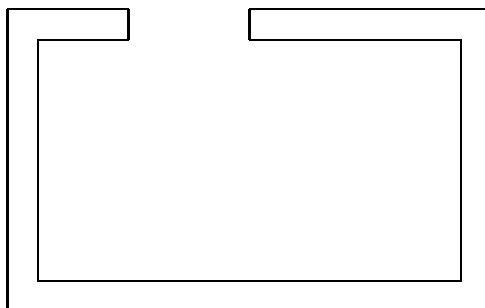
Można tworzyć własne liczniki, a także wpływać na ich wartości, służą do tego następujące polecenia:

- `\newcounter{nowy_licznik}[stary_licznik]` – definiowanie nowego licznika, opcjonalny parametr pozwala na powiązanie go z już istniejącym – kiedy stary licznik zostanie zwiększony, nowy zostanie wyzerowany;
- `\setcounter{nazwa}{wartość}` – ustala wartość licznika,
- `\addtocounter{nazwa}{wartość}` – dodaje wartość do licznika (dopuszczalne są wartości ujemne).

3.2. Zastosowania wymagające obliczeń. Braki arytmetyczne systemu $\text{\LaTeX}$

System $\text{\TeX}$ powstał z myślą o składzie tekstów matematycznych, czy też zagadnień programistycznych, jednak nie udostępnia użytkownikowi narzędzi do obliczeń arytmetycznych. Jak pokazano, użytkownik ma tylko pośredni wpływ na obliczenia dokonywane przez system i nie ma możliwości wykorzystać arytmetyki systemu do obliczeń numerycznych.

Braki te są najbardziej odczuwalne przy programowaniu grafiki wektorowej. Za przykład posłuży plan prostego pomieszczenia przedstawiony na rysunku 3.1. Jak widać rysunek składa się z 12 linii określonej długości umieszczonych w odpowiednim miejscu obszaru rysowania.



Rysunek 3.1. Plan pomieszczenia wykonany w $\text{\LaTeX}$ -u
Źródło: opracowanie własne

Linie w $\text{\LaTeX}$ -u definiuje się komendą: `\line(x,y){d}`, gdzie x i y są współrzędnymi wektora kierunkowego, określającego zwrot linii, natomiast d oznacza jej długość. Aby umieścić linię w wybranym miejscu używa się polecenia `\put(x,y){element}`, które umieszcza wybrany element na pozycji określonej współrzędnymi x i y . Współrzędne te zależą od kontekstu w jakim zostaną użyte – określają pozycję względem np. strony, wyrazu lub, jak w tym przypadku, środowiska *picture*. Kod źródłowy rysunku 3.1 wygląda następująco:

```
\begin{picture}(80,50)
  \put(0,50){\line(1,0){20}} % Północna ściana
  \put(5,45){\line(1,0){15}} % Lewa część ściany
  \put(20,45){\line(0,1){5}}
  \put(40,50){\line(1,0){40}} % Prawa część ściany
  \put(40,45){\line(1,0){35}}
  \put(40,45){\line(0,1){5}}
  \put(0,0){\line(0,1){50}} % Zachodnia ściana
  \put(5,5){\line(0,1){40}}
  \put(80,0){\line(0,1){50}} % Wschodnia ściana
  \put(75,5){\line(0,1){40}}
  \put(0,0){\line(1,0){80}} % Południowa ściana
  \put(5,5){\line(1,0){70}}
\end{picture}
```

Chcąc po stworzeniu rysunku dokonać w nim zmiany np. przesunąć drzwi o 5 jednostek, w stronę wschodniej ściany, należy poprawić definicję odpowiednich elementów – w tym wypadku sześciu linii:

```

\begin{picture}(80,50)
\put(0,50){\line(1,0){25}} % Północna ściana
\put(5,45){\line(1,0){20}} % Lewa część ściany
\put(25,45){\line(0,1){5}}
\put(45,50){\line(1,0){35}} % Prawa część ściany
\put(45,45){\line(1,0){30}}
\put(45,45){\line(0,1){5}}
(...)
\end{picture}

```

Jak widać wprowadzanie nawet, wydawałoby się, drobnych poprawek jest dość kłopotliwe. Sytuacja komplikuje się jeszcze bardziej przy skomplikowanych ilustracjach. Problem ten rozwiązałyby możliwość określenia relacji pomiędzy wszystkimi, lub niektórymi, wymiarami np.:

```

\begin{picture}(szer,dług)
\put(0,dług){\line(1,0){a}}
\put(g,dług-g){\line(1,0){a-g}}
\put(a,dług-g){\line(0,1){g}}
\put(a,dług){\line(1,0){szer-a-b}}
\put(a,dług-g){\line(1,0){szer-a-b-g}}
\put(a,dług-g){\line(0,1){g}}
\put(0,0){\line(0,1){dług}}
\put(g,g){\line(0,1){dług-2*g}}
\put(szer,0){\line(0,1){dług}}
\put(szer-g,g){\line(0,1){dług-2*g}}
\put(0,0){\line(1,0){szer}}
\put(g,g){\line(1,0){szer-2*g}}
\end{picture}

```

Gdzie:

- **a** – szerokość ściany po lewej stronie otworu,
- **b** – szerokość otworu na drzwi,
- **g** – grubość ściany,
- **szer** – szerokość pomieszczenia,
- **dług** – długość pomieszczenia.

Taką definicję rysunku można by było wykorzystać do utworzenia planu omawianego pokoju o dowolnie zdefiniowanych wymiarach²⁾, jednak $\text{\LaTeX}$ nie pozwala na przechowywanie zmiennych liczbowych i wykonywanie na nich obliczeń.

²⁾ Oczywiście w granicach rozsądku i geometrycznej poprawności; np. grubość ścian, przy takiej definicji, nie może przekraczać połowy mniejszego z wymiarów pokoju.

3.3. Propozycja uzupełnienia systemu

Jak wykazano w systemie $\text{\LaTeX}$ – pomimo, że jest kompletnym narzędziem składu tekstu – istnieją możliwości, których funkcjonalność nie jest w pełni zapewniona. Istnieją także zastosowania, których nie da się w pełni wykorzystać bez posługiwania się kalkulatorem czy dodatkowym oprogramowaniem. Wynika to z faktu, że pomimo swojego matematycznego przeznaczenia, w $\text{\LaTeX}$ -u nie ma możliwości wykonywania obliczeń innych niż zmiany licznika, czy przeskalowanie odległości. Istnieją zewnętrzne programy generujące kod źródłowy, który można wkleić do dokumentu, ale w samym systemie nie ma możliwości obliczania wyrażeń ani implementacji algorytmu.

Jako, że $\text{\LaTeX}$ jest dość skomplikowanym systemem i wbudowywanie w niego dodatkowych narzędzi obliczeniowych czy programistycznych mogłoby wprowadzić zamieszanie, w pracy tej zaproponowano koncepcję uzupełnienia tej funkcjonalności za pomocą osobnego systemu – preprocesora arytmetycznego.

System ten miał obejmować kompilator i język umożliwiający programowanie obliczeń w wydzielonych blokach pliku źródłowego $\text{\LaTeX}$ -a. Kompilator systemu miał dokonywać pierwszego przetwarzania takiego pliku źródłowego, wykonać instrukcje i wygenerować na wyjściu „czysty” kod $\text{\LaTeX}$ -a. Działanie tego systemu można zatem porównać do zasady działania języka *PHP*, gdzie przy wywołaniu skryptu wykonywane są instrukcje programowe i generowany jest plik *HTML*.

Preprocesor postanowiono nazwać od pierwszej litery imienia autora – KTeX . Następny rozdział został poświęcony opisowi realizacji tego systemu, wraz z przedstawieniem dostępnych w nim funkcji.

4. Preprocesor arytmetyczny – K_TE_X

4.1. Koncepcja wykonania

Jak wspomniano w 3.3, preprocesor arytmetyczny dla systemu L_AT_EX ma wykorzystywać własny język i składnię, w formie „wtrąceń”, w pliku źródłowym. Aby odróżnić pliki źródłowe, z kodem K_TE_X-a, od plików L_AT_EX-a, przyjęto dla nich rozszerzenie: **ktex**. Składnię języka postanowiono bazować do pewnego stopnia na idei składni *PHP*. Bloki instrukcji przyjęto oznaczać symbolami: **<#** – jako początek bloku i **#>** – koniec. Kolejne instrukcje, jak w większości języków programowania, przyjęto oddzielać średnikiem.

Głównym zadaniem preprocesora K_TE_X miało być wykonywanie obliczeń arytmetycznych zarówno na liczbach rzeczywistych jak i całkowitych. Do implementacji przewidziano podstawowe działania arytmetyczne – dodawanie, mnożenie i dzielenie, dzielenie modulo, potęgowanie i pierwiastkowanie. Wyniki obliczeń, aby dało je się wykorzystać, muszą dać się przechowywać w pamięci – system miał więc zapewnić deklarowanie zmiennych.

W czasie pracy nad systemem rozważana była implementacja specyficznych działań i funkcji, takich jak np. wartość bezwzględna, podłoga, sufit, czy też logarytm lub sinus. Zamiast tego postanowiono umożliwić użytkownikowi systemu zaprogramowanie własnych funkcji – wprowadzono podstawowe konstrukcje programistyczne pozwalające implementować algorytmy w języku K_TE_X. Mowa tu o deklarowaniu funkcji, tworzeniu pętli i instrukcji warunkowych. Wprowadzenie instrukcji warunkowych wymagało wprowadzenia operacji porównywania liczb, zaimplementowano także działania logiczne: koniunkcję, alternatywę i negację.

Do realizacji preprocesora wybrano język *Java2SE*. Głównymi zaletami tego języka jest przenośność na dowolną platformę systemową, na której zainstalowano *wirtualną*

maszynę Java¹⁾. Język ten pozwala także na sprawne przetwarzanie łańcuchów znakowych dzięki rozbudowanej bazie metod²⁾ klasy *String* – czyli funkcji operujących na łańcuchach znakowych. Bardzo ważną zaletą języka okazała się też obsługa przechwytywania wyjątków, wykorzystana do komunikowania użytkownikowi błędów składni.

W kolejnych podrozdziałach przedstawiono sposób realizacji wymienionych funkcji systemu. W niektórych podrozdziałach podano numery linii, pliku źródłowego *Dokumentu prezentacyjnego systemu K_TE_X*, zawierające przykłady użycia składni i dołączonego w załączniku B. Załącznik C zawiera kod źródłowy dokumentu L^AT_EX-a wygenerowany z tego pliku, natomiast załącznik D – gotowy, skompilowany dokument.

4.2. Realizacja systemu K_TE_X

Działanie programu K_TE_X polega na przetwarzaniu pliku źródłowego (*.ktex*) – czyli wyszukaniu bloków instrukcji i zastąpieniu ich wynikami działania kodu. W programie nie uwzględniono graficznego interfejsu, uruchomienia – kompilacji dokonuje się wywołując program z odpowiednimi parametrami:

```
java ktex nazwa-pliku-źródłowego.ktex nazwa-pliku-wynikowego.tex
```

Podanie nazwy pliku wynikowego jest opcjonalne – domyślnie utworzony zostanie plik *out.tex*.

Program K_TE_X można podzielić na trzy części zadaniowe:

1. Metoda *analiza* – główna część programu, do jej zadań należy:
 - czytanie pliku wejściowego – głównego pliku źródłowego lub pomocniczych (odpowiedzialnych za pętle i funkcje),
 - analiza składni – poszukiwanie bloków instrukcji i deklaracji funkcji (wraz z obsługą; tylko podczas przetwarzania głównego pliku – nie można deklarować funkcji np. wewnątrz pętli), pętli (przekazanie obsługi do osobnej metody) oraz instrukcji warunkowych (wraz z obsługą);

¹⁾ ang. *Java Virtual Machine* – wchodzi w skład *Java Runtime Environment* (w skrócie *JRE*), środowiska uruchamiania aplikacji, stanowiącego specyficzny przykład wirtualizacji. Dla konkretnych systemów operacyjnych wydawane są dedykowane wersje *JRE*, które umożliwiają wykonanie programu – w tym symulowanym środowisku – z jednakowym wynikiem.

²⁾ Metodą w języku *Java* – i ogólnie w programowaniu obiektowym – jest funkcja działająca na rzecz klasy, w której została zadeklarowana.

- przekazanie znalezionych instrukcji do dalszej analizy i parsingu,
 - przepisywanie wyników do pliku wyjściowego – głównego lub pomocniczego.
2. Metoda *analizaFor* – osobna metoda służąca do obsługi pętli *for* (szczegółowy opis działania w 4.2.5).
 3. Metody analizy składni – dokonują wstępnego parsingu instrukcji i parsingu wyrażeń arytmetycznych, wraz z obsługą zmiennych, liczb i wywołań funkcji (podrozdział 4.2.3).

Po uruchomieniu programu z pliku źródłowego zaczytywane są kolejne linie, w których poszukiwane są znaczniki początku bloku instrukcji (<#). Do ich wyszukiwania – jak też do analizy wierszy w innych miejscach – posłużyły autorowi następujące metody, którym jako parametr podaje się poszukiwany ciąg znaków:

- `contains()` – metoda zwraca wartość *true* jeśli łańcuch na rzecz którego została wywołana, zawiera łańcuch będący argumentem wywołania,
- `indexOf()` – j/w, ale zwraca pozycję pierwszego wystąpienia tego łańcucha (jeśli nie zostanie odnaleziony zwraca -1),
- `lastIndexOf()` – j/w, ale zwraca pozycję ostatniego wystąpienia.

Jeśli w danej linii nie zostanie odnaleziony początek bloku instrukcji, zostaje ona przepisana do pliku wynikowego bez zmian. W przypadku odnalezienia znaków <#, następuje jeszcze dalsza analiza kontekstu tego symbolu, ponieważ specyficznym przypadkiem jego użycia jest rozpoczęcie deklaracji funkcji, pętli lub instrukcji warunkowej (sytuacje takie zostaną omówione w dalszej części pracy). Jeżeli potwierdzono, że odnaleziony symbol oznacza zwykły blok języka K_TE_X, to następuje pobranie i analiza kolejnych instrukcji (kończących się średnikiem), aż do napotkania znaków końca bloku instrukcji (#>). Jeśli koniec bloku nie występuje w tym samym wierszu, wczytywany jest kolejny i analizowane są występujące w nim instrukcje, co powtarza się do znalezienia końca bloku.

Wszystkie instrukcje K_TE_X-a można podzielić na takie, które coś „wypisują” – dają wynik (w postaci tekstu lub liczby) w pliku docelowym, oraz na „ciche” – zmieniające

wartości zadeklarowanych zmiennych, ale nie dające tekstu w pliku wynikowym. Podstawowe instrukcje to:

- przypisanie – w ich wyniku zmieniona zostaje wartość zmiennej, nic nie jest wypisywane,
- działania arytmetyczne – jeśli nie są użyte do przypisania zostanie wypisany wynik takiego działania,
- wywołania funkcji – mogą wypisywać tekst, liczbę i/lub zmieniać wartości zmiennych (także deklarować nowe zmienne).

Do przypisania, poza zwykłym zapisem $a = b$, można wykorzystać cztery operatory skróconego zapisu, aktualizacji zmiennej: $a+=b$, $a-=b$, $a*=b$ i $a/=b$, czyli odpowiednio np. dodanie zmiennej a do zmiennej b i zapisanie wyniku w zmiennej a . Przykład przypisania w linii 40 załącznika B.

Wprowadzono też o możliwość komentowania kodu (znakami `//`) działającego na takiej samej zasadzie jak `%` w $\text{\LaTeX}$ -u – komentarz usuwa przejście do nowej linii.

4.2.1. Gramatyka $\text{K}\text{\TeX}$ -a

Działanie programu, jak i struktura języka $\text{K}\text{\TeX}$, opiera się na gramatyce bezkontekstowej. Gramatyka bezkontekstowa opisuje ciąg kroków, jakie należy wykonać, aby zbudować słowo należące do danego języka – w tym wypadku słowem jest poprawny kod pliku źródłowego. Poprawnie napisana gramatyka pozwala np. określić, które działanie ma pierwszeństwo wykonania, aby otrzymać poprawny wynik. Autor opierał się w pracy nad programem i gramatyką teorią znaną z przedmiotu [9].

Gramatyka bezkontekstowa składa się z:

- alfabetu terminalnego, czyli znaków jakie wchodzi w skład poprawnego słowa – dla $\text{K}\text{\TeX}$ -a będą to praktycznie wszystkie znaki jakich można używać w $\text{\LaTeX}$ -u,
- alfabetu nieterminalnego, czyli zbioru pośrednich elementów na drodze do otrzymania słowa – w kodzie systemu odzwierciedlone są odpowiednim algorytmem lub rekurencyjną metodą (o nazwie pokrywającej się z danym nieterminalem),
- skończonego zbioru produkcji – przepisów przejścia z nieterminala do słowa pośredniego (ciągu zawierającego same nieterminale lub zarówno nieterminale i terminale)

lub właściwego słowa, należącego do języka (ciągu samych tylko terminali); symbol $\rightarrow$ oznacza pojedynczą produkcję, natomiast $::=$ oznacza zapis skrótowy (kilka produkcji w jednej linii – oddzielone kreską pionową),

- wybranego nieterminala początkowego, od którego zaczyna się wywód – tworzenie słowa; w gramatyce K_TE_X-a jest nim: $\langle \text{źródło} \rangle$,
- słowa pustego oznaczanego symbolem λ .

$\langle \text{źródło} \rangle$	$\rightarrow$	λ
$\langle \text{źródło} \rangle$	$\rightarrow$	$\langle \text{źródło} \rangle \langle \text{treść} \rangle$
$\langle \text{źródło} \rangle$	$\rightarrow$	$\langle \text{źródło} \rangle \langle \text{komentarz} \rangle$
$\langle \text{źródło} \rangle$	$\rightarrow$	$\langle \text{źródło} \rangle \langle \text{kod} \rangle$
$\langle \text{treść} \rangle$	$\rightarrow$	„dowolny tekst z pominięciem par znaków: $\langle \# \ i \ \# \rangle$ ”
$\langle \text{komentarz} \rangle$	$\rightarrow$	// „dowolny tekst zakończony znakiem końca linii”
$\langle \text{kod} \rangle$	$\rightarrow$	$\langle \text{deklaracja funkcji}^* \rangle$
$\langle \text{kod} \rangle$	$\rightarrow$	$\langle \text{blok} \rangle$
$\langle \text{blok} \rangle$	$\rightarrow$	$\langle \# \langle \text{instrukcje} \rangle \# \rangle$
$\langle \text{blok} \rangle$	$::=$	$\langle \text{for}^* \rangle \mid \langle \text{if}^* \rangle$
$\langle \text{kod wewnętrzny} \rangle$	$\rightarrow$	λ
$\langle \text{kod wewnętrzny} \rangle$	$\rightarrow$	$\langle \text{kod wewnętrzny} \rangle \langle \text{treść} \rangle$
$\langle \text{kod wewnętrzny} \rangle$	$\rightarrow$	$\langle \text{kod wewnętrzny} \rangle \langle \text{blok} \rangle$
$\langle \text{instrukcje} \rangle$	$::=$	$\lambda \mid \langle \text{instrukcje} \rangle \langle \text{instrukcja} \rangle$;
$\langle \text{instrukcja} \rangle$	$\rightarrow$	$\langle \text{przypisanie} \rangle$
$\langle \text{instrukcja} \rangle$	$\rightarrow$	$\langle \text{in/de-krementacja} \rangle$
$\langle \text{instrukcja} \rangle$	$\rightarrow$	$\langle \text{wyrażenie}^* \rangle$
$\langle \text{przypisanie} \rangle$	$\rightarrow$	$\langle \text{zmienna} \rangle \langle \text{operator przypisania} \rangle \langle \text{wyrażenie} \rangle$
$\langle \text{in/de-krementacja} \rangle$	$::=$	$\langle \text{zmienna} \rangle ++ \mid \langle \text{zmienna} \rangle --$
$\langle \text{zmienna}^* \rangle$	$\rightarrow$	$\& \langle \text{nazwa} \rangle$
$\langle \text{nazwa} \rangle$	$\rightarrow$	„ciąg liter i cyfr”
$\langle \text{operator przypisania} \rangle$	$::=$	$= \mid += \mid -= \mid *= \mid /=$

Rysunek 4.1. Podstawowa gramatyka języka K_TE_X

Źródło: opracowanie własne

Rysunek 4.1 przedstawia podstawową gramatykę języka K_TE_X. Nieterminale oznaczono symbolami: $\langle \ i \ \rangle$. Nieterminal $\langle \text{źródło} \rangle$ oznacza cały plik wejściowy, kolejne nieterminale odpowiadają zaś poszczególnym składowym kodu K_TE_X-a lub tekstowi. W kontekście kodu K_TE_X-a przez *tekst* rozumiane są także wszystkie komendy L_AT_EX-a.

Gramatyka wyrażeń arytmetycznych, a także elementy, takie jak deklaracja funkcji, pętle i instrukcje warunkowe, zostały przedstawione przy omawianiu tych konstrukcji – oznaczono to symbolem $*$ przy nazwach nieterminali. Nieterminal $\langle \text{kod wewnętrzny} \rangle$

wykorzystywany będzie w produkcjach pochodnych od nieterminali $\langle if \rangle$, $\langle for \rangle$ i $\langle deklaracja\ funkcji \rangle$. Użycie tego dodatkowego nieterminala pozwala tworzyć pętle i instrukcje warunkowe w innych pętlach, instrukcjach warunkowych i deklaracjach funkcji, ale nie pozwala deklarować funkcji wewnątrz innych deklaracji, czy też wewnątrz pętli lub instrukcji warunkowych. Zestawienie całej gramatyki K_TE_X-a znajduje się w załączniku A.

4.2.2. Zmienne

W programie – dla uproszczenia obsługi systemu – postanowiono zrezygnować z jawnej deklaracji typów zmiennych. Wszystkie zmienne są deklarowane w momencie pierwszego przypisania im wartości i są przechowywane jako zmienne globalne, typu zmiennoprzecinkowego – *float*. Jednak, dla wyników w postaci liczb całkowitych, funkcjonuje „drukowanie” ich wartości bez części ułamkowej (czyli bez „.0”).

Do przechowywania pojedynczej zmiennej wykorzystano osobną klasę. Obiekt tej klasy stanowi reprezentację zmiennej i w odpowiednich polach przechowuje jej identyfikator (pole *nazwa* – typu *String*) i wartość liczbową (pole *wartość* – typu *float*). Natomiast do przechowywania kolejnych zmiennych (instancji klasy *Zmienna*) użyto obiektu *ArrayList* – tablicy (lub też listy) o zmiennej długości, dla której nie ma konieczności zadawania z góry maksymalnego rozmiaru. Taki obiekt pozwala na przechowywanie teoretycznie dowolnej liczby elementów.

Podczas wczesnych testów programu K_TE_X, pojawił się problem związany z konwersją typu *String* na *float* – błąd konwersji, pojawiający się na końcu zakresu typu *float* (np. przy konwersji łańcucha: "1" na typ *float* otrzymywano wartość: 1.00000001). Aby rozwiązać ten problem wszystkie konwersje i obliczenia wewnątrz programu wykonywane są na pomocniczych zmiennych podwójnej precyzji (*double*), które następnie zostają rzutowane na typ *float*. Przy konwersji z użyciem typu *double* nadal pojawia się wspomniany błąd, zostaje on jednak „obcinany” przy rzutowaniu i nie ma wpływu na poprawność obliczeń.

Identyfikatory zmiennych przyjęto oznaczać znakiem: $\&$. Pozwoliło to ograniczyć porównania przy analizie składni (zamiast sprawdzenia czy kolejny znak znajduje się w zakresie od *a* do *z*, sprawdzane jest czy pierwszy równa się $\&$), oraz dało możliwość

rozpoznawania po jednym znaku, czy dany element instrukcji jest liczbą, zmienną lub wywołaniem funkcji.

W K_TE_X-u zaimplementowano także zmienne indeksowane (tablicowe) o dowolnym zakresie. Zmienne te tworzy się używając dowolnej liczby par nawiasów kwadratowych w nazwie zmiennej oraz argumentów w nawiasach oddzielonych przecinkami. Dopuszczalne formy deklaracji takiej zmiennej, jak i odwołania do niej mają postać np.:

- `&zmienna[1],`
- `&zmienna[7][&i][13][&i+&j],`
- `&zmienna[&i,&j],`
- `&zmienna[&t][&x,&y,&z],`
- `&zmienna[&z]numerem.`

Mechanika, tworzenia zmiennych tablicowych, opiera się na nadawaniu identyfikatorów zawierających w sobie nawiasy kwadratowe wraz z wartościami liczbowymi i przechowywanie ich, w takiej postaci, we wspomnianym polu *nazwa*, klasy *Zmienna*. Tworzenie identyfikatorów zmiennych tablicowych opisuje gramatyka przedstawiona na rysunku 4.2 – jest to uzupełnienie produkcji wynikających z nieterminała $\langle zmienna \rangle$. Wartości w nawiasach obliczane są tą samą metodą co argumenty wywołania funkcji, co zostało omówione w 4.2.6. Przykłady użycia zmiennych znajdują się w załączniku B w linii 59.

$$\begin{array}{ll}
 \langle zmienna \rangle & \rightarrow \&\langle nazwa \rangle \langle zmienna \ tab. \rangle \\
 \langle zmienna \ tab. \rangle & ::= \lambda \mid \langle tablica \rangle \mid \langle zmienna \ tab. \rangle \langle tablica \rangle \langle nazwa \rangle \\
 \langle tablica \rangle & ::= [\langle argumenty \rangle] \mid \langle tablica \rangle [\langle argumenty \rangle] \\
 \langle argumenty \rangle & ::= \langle wyrażenie \rangle \mid \langle argumenty \rangle, \langle wyrażenie \rangle \\
 \langle nazwa \rangle & \rightarrow \text{„ciąg liter i cyfr”}
 \end{array}$$

Rysunek 4.2. Gramatyka tworzenia zmiennych tablicowych
Źródło: opracowanie własne

4.2.3. Obliczenia arytmetyczne. Gramatyka wyrażeń języka K_TE_X

W K_TE_X-u dostępne są następujące działania arytmetyczne (w nawiasie podano operator używany w języku):

- dodawanie (+) i odejmowanie (−), także w postaci operatorów inkrementacji (`&zmienna++`) i dekrementacji (`&zmienna--`),

- mnożenie ($*$) i dzielenie ($/$),
- dzielenie modulo ($\%$) – reszta z dzielenia całkowitego,
- potęgowanie ($\wedge$) i pierwiastkowanie – dla potęgi: $|n| < 1$.

Przykłady działań arytmetycznych znajdują się w linii 49 załącznika B.

Aby zapewnić poprawność wykonywanych obliczeń (kolejność wykonywania i łączność działań) posłużono się stworzoną w tym celu gramatyką języka wyrażeń arytmetycznych. Wykorzystano ją następnie do napisania parsera. W tej gramatyce słowem jest poprawne wyrażenie arytmetyczne, natomiast alfabetem terminalnym są litery, cyfry, znaki działań arytmetycznych, przecinek (separator argumentów wywołania funkcji), kropka (separator części ułamkowej liczby), nawiasy, oraz znaki oznaczające zmienne ($\&$) oraz funkcje ($_$). Rysunek 4.3 przedstawia produkcje definiujące tę gramatykę jaka posłużyła do zaprogramowania parsera wyrażeń arytmetycznych. Nazwy nieterminali pokrywają się z nazwami metod w kodzie K_TE_X-a. Każda taka metoda jako parametr pobiera ciąg znaków – część słowa do analizy. Zanalizowany kod zostaje wykonany – pobrana zostaje wartość zmiennej lub wykonane działanie i zwrócony zostaje wynik, także w postaci łańcucha znaków.

$\langle \text{wyrażenie} \rangle$	$\rightarrow$	$\langle \text{wyrażenie} \rangle + \langle \text{składnik} \rangle$
$\langle \text{wyrażenie} \rangle$	$\rightarrow$	$\langle \text{wyrażenie} \rangle - \langle \text{składnik} \rangle$
$\langle \text{wyrażenie} \rangle$	$\rightarrow$	$\langle \text{składnik} \rangle$
$\langle \text{składnik} \rangle$	$\rightarrow$	$\langle \text{składnik} \rangle * \langle \text{czynnik} \rangle$
$\langle \text{składnik} \rangle$	$\rightarrow$	$\langle \text{składnik} \rangle / \langle \text{czynnik} \rangle$
$\langle \text{składnik} \rangle$	$\rightarrow$	$\langle \text{składnik} \rangle \% \langle \text{czynnik} \rangle$
$\langle \text{składnik} \rangle$	$\rightarrow$	$\langle \text{czynnik} \rangle$
$\langle \text{czynnik} \rangle$	$\rightarrow$	$\langle \text{atom} \rangle \wedge \langle \text{czynnik} \rangle$
$\langle \text{czynnik} \rangle$	$\rightarrow$	$\langle \text{atom} \rangle$
$\langle \text{atom} \rangle$	$\rightarrow$	$(\langle \text{wyrażenie} \rangle)$
$\langle \text{atom} \rangle$	$\rightarrow$	$\langle \text{pobranie zmiennej} \rangle$
$\langle \text{atom} \rangle$	$\rightarrow$	$\langle \text{wywołanie funkcji}^* \rangle$
$\langle \text{atom} \rangle$	$\rightarrow$	$\langle \text{liczba} \rangle$
$\langle \text{pobranie zmiennej} \rangle$	$\rightarrow$	$\langle \text{zmienna} \rangle$
$\langle \text{liczba} \rangle$	$\rightarrow$	„ciąg cyfr (opcjonalnie z kropką i/lub minusem)”

Rysunek 4.3. Gramatyka wyrażeń języka K_TE_X

Źródło: opracowanie własne

Poprawną kolejność wykonywania działań, zapewnia kolejność nieterminali w ciągu produkcji – im bliżej nieterminala początkowego, tym działanie ma mniejszy priorytet

– wykonane zostanie później. Zapewnienie łączności działań jest realizowane przez odpowiednią pozycję nieterminali w produkcjach rekurencyjnych np.:

- produkcja: $\langle \textit{wyrażenie} \rangle \rightarrow \langle \textit{wyrażenie} \rangle + \langle \textit{składnik} \rangle$
zapewnia łączność lewostronną działania dodawania,
- produkcja: $\langle \textit{czynnik} \rangle \rightarrow \langle \textit{atom} \rangle \sim \langle \textit{czynnik} \rangle$
zapewnia łączność prawostronną potęgowania.

4.2.4. Instrukcja warunkowa i operacje logiczne

Składnia instrukcji warunkowej K_TE_X-a ma postać:

```
<#if{wyrażenie warunkowe}
...
<# ... instrukcje, for, if ... #>
... tekst ...
if#> <#else
...
<# ... instrukcje, for, if ... #>
... tekst ...
else#>
```

Instrukcja *if* stanowi niezależny blok, w którym występują właściwe bloki instrukcji. W przypadku, gdy całe wyrażenie warunkowe zwróci wartość *fałsz*, wszystkie instrukcje i tekst wewnątrz bloku zostaną pominięte przez kompilator, natomiast wykonana zostanie zawartość bloku *else* – o ile taki istnieje. Sposób deklaracji instrukcji *if* opisuje gramatyka przedstawiona na rysunku 4.4. Nieterminal $\langle \textit{kod wewnętrzny} \rangle$ wprowadzony został przy podstawowej gramatyce (rysunek 4.1).

$$\begin{aligned} \langle \textit{if} \rangle &\rightarrow \text{<\#if\{<wyrażenie log.*>\}} \\ &\quad \langle \textit{kod wewnętrzny} \rangle \\ &\quad \text{if\#> } \langle \textit{else} \rangle \\ \langle \textit{else} \rangle &\rightarrow \lambda \\ \langle \textit{else} \rangle &\rightarrow \text{<\#else} \\ &\quad \langle \textit{kod wewnętrzny} \rangle \\ &\quad \text{else\#>} \end{aligned}$$

Rysunek 4.4. Gramatyka instrukcji *if*
Źródło: opracowanie własne

W systemie zaimplementowano operacje logiczne, jednak nie występują tu zmienne typu logicznego. Przyjęto wartość liczbową „0” traktować jako logiczny *fałsz*, a każdą

inna wartość jako *prawdę*. Przeznaczeniem operacji logicznych jest ich wykorzystanie w instrukcjach warunkowych i w tej wersji systemu nie uwzględniono możliwości innego ich użycia. Podstawową operacją logiczną jest porównanie, w K_TE_X-u dostępne są następujące typy porównań:

- równy (`==`),
- równy lub większy (`=>`),
- równy lub mniejszy (`=<`),
- większy (`>`),
- mniejszy (`<`),
- różny (`<>`).

Operacje porównania można łączyć w wyrażenia logiczne za pomocą operatorów logicznych:

- koniunkcji (`&&`),
- alternatywy (`||`),
- negacji (`!`).

Przykłady użycia instrukcji *if* znajdują się w linii 88 załącznika B.

Na potrzeby operacji logicznych rozszerzono gramatykę o produkcje przedstawione na rysunku 4.5. Nieterminal *⟨logiczny⟩* odpowiada pojedynczemu porównaniu. Operatory logiczne korzystają z tej samej gramatyki co operatory arytmetyczne – koniunkcja traktowana jest jak mnożenie, natomiast alternatywa jako dodawanie. Nieterminal *⟨wyrażenie log.⟩*, *⟨składnik log.⟩* i *⟨czynnik log.⟩* odpowiadają analogicznym nieterminalom i ich metodom z gramatyki wyrażeń K_TE_X-a – w kodzie programu dokonana zostaje zamiana operatorów logicznych na arytmetyczne.

Alfabet terminali rozszerzony został o znak negacji (`!`) oraz znaki, z których składają się operatory porównań (`>`, `<`, `=`). Operacje porównania nie mogą być łączone – możliwe jest użycie tylko jednego operatora w porównaniu (nie można stosować konstrukcji typu: `a > b < c`, zamiast tego należy użyć: `a > b && b < c`).

$$\begin{aligned}
\langle \text{wyrażenie log.} \rangle &\rightarrow \langle \text{wyrażenie log.} \rangle | \langle \text{składnik log.} \rangle \\
\langle \text{składnik log.} \rangle &\rightarrow \langle \text{składnik log.} \rangle \& \langle \text{czynnik log.} \rangle \\
\langle \text{czynnik log.} \rangle &\rightarrow \langle \text{logiczny} \rangle \\
\langle \text{logiczny} \rangle &\rightarrow ! \langle \text{logiczny} \rangle \\
\langle \text{logiczny} \rangle &\rightarrow \langle \text{wyrażenie} \rangle \langle \text{operator porównania} \rangle \langle \text{wyrażenie} \rangle \\
\langle \text{operator porównania} \rangle &::= == | => | =< | <> | > | <
\end{aligned}$$

Rysunek 4.5. Gramatyka wyrażeń logicznych

Źródło: opracowanie własne

4.2.5. Pętla *for*

Składnia pętli *for* w KTeX-u ma postać:

```

<#for{&nazwa-zmiennej-sterującej : od : krok : do}
...
<# ... instrukcje, for, if ... #>
... tekst ...
for#>

```

Zdefiniowanie pętli polega na wybraniu nazwy zmiennej sterującej, określeniu jej zakresu (pola *od* i *do*) oraz kroku z jakim ma być zmieniana. Parametry zakresu jak i krok zmiennej, przyjęto w programie jako zmienne typu *float*. Dzięki temu można stworzyć dowolną wariację pętli, a odpowiednio nadając parametry i dodając wewnątrz pętli instrukcję warunkową, można zbudować pętlę o zachowaniu identycznym z pętlami typu *while* (przykłady w linii 115 i 126 załącznika B). Możliwe jest także używanie w polach *od*, *do* i *krok* wyrażeń (włącznie z wywołaniami funkcji). Odpowiednią gramatykę przedstawiono na rysunku 4.6.

$$\begin{aligned}
\langle \text{for} \rangle &\rightarrow \text{<\#for\{\&\langle \text{nazwa} \rangle : \langle \text{wyrażenie} \rangle : \langle \text{wyrażenie} \rangle : \langle \text{wyrażenie} \rangle\}} \\
&\quad \langle \text{kod wewnętrzny} \rangle \\
&\text{for\#>}
\end{aligned}$$

Rysunek 4.6. Gramatyka pętli *for*

Źródło: opracowanie własne

Obsługą pętli zajmuje się metoda *analizaFor*, która zostaje wywołana, kiedy główna metoda napotka w kodzie początek pętli. W metodzie tej następuje interpretacja deklaracji, następnie kod zawarty wewnątrz pętli zostaje przepisany do pliku pomocniczego.

Pliki te tworzone są w katalogu `tmp`, ich nazwy mają postać: `in#.tmp`, gdzie `#` oznacza poziom zagłębienia pętli. Plik taki zostaje przekazany do analizy, za pomocą głównej metody – *analiza*. Jest on więc poddawany, w ten sam sposób co główny plik

źródłowy, całemu procesowi parsingu, włącznie z analizą kolejnych pętli – spowoduje to wywołanie kolejny raz metody *analizaFor* jednak z innym numerem zagłębienia. W analizie kodu pętli wyłączono możliwość deklaracji funkcji i przekierowano wyjście do osobnego pliku pomocniczego – `out#.tmp`.

Kod pętli zostaje wykonany odpowiednią ilość razy, zgodnie z definicją, dzięki ciągłej kontroli wartości zmiennej sterującej K_TE_X-a – dopuszcza to możliwość zmiany tej zmiennej wewnątrz pętli (i realizację np. wspomnianej konstrukcji pętli *while*).

Po wykonaniu pętli wynik zostaje przepisany z pliku pomocniczego do pliku wynikowego i proces przetwarzania pliku źródłowego jest kontynuowany.

4.2.6. Deklaracja i wywołanie funkcji

Składnia deklaracji funkcji ma postać:

```
<#function:nazwa-funkcji{&argument1, &argument2, ...}  
...  
  <# ... instrukcje, for, if ... #>  
  ... tekst ...  
function#>
```

W systemie postanowiono ograniczyć się do jednej ogólnej deklaracji funkcji, jednak w zależności od kodu samej funkcji, może ona działać jako:

- standardowa funkcja – zwracająca wynik (liczbę) dla podanych argumentów – jedynie tak przygotowaną funkcję można stosować w wyrażeniach, warunkach, deklaracjach pętli oraz jako argumenty innych wywołań funkcji, należy jednak zwrócić uwagę na kolejność wykonywanych operacji – ponieważ wszystkie zmienne w K_TE_X-u są globalne, wykonanie funkcji może wpłynąć na wartości używanych zmiennych, więc i na wynik całego wyrażenia;
- procedura – nic nie zwracająca ale mogąca wykonywać obliczenia, zmieniać wartości zmiennych (także deklarować nowe zmienne),
- makro – generujące, w zależności od argumentów, dowolną ilość tekstu (czyli też komend L^AT_EX-owych i obliczonych wyrażen K_TE_X-a) w pliku wynikowym.

Przykłady tych użyć znaleźć można w liniach 154, 171 i 181, w załączniku B.

Obsługą deklaracji funkcji zajmuje się metoda *analiza*. Zadeklarowana funkcja przechowywana jest w postaci listy argumentów i kodu samej funkcji. Lista argumentów

jest obiektem, omówionej przy okazji zmiennych w 4.2.2, klasy *ArrayList* i jest przechowywana w obiekcie klasy *Funkcja*. Konstruktor tej klasy wypełnia tę listę nazwami argumentów (o ile takie istnieją w deklaracji) – deklaruje odpowiednie zmienne K_TE_X-owe. Kod funkcji przechowywany jest natomiast w pliku, w katalogu `tmp`, o nazwie `nazwa_funkcji.fun`.

Funkcję wywołuje się, w bloku instrukcji, w następujący sposób:

```
_nazwa-funkcji[argument1, argument2, ...]
```

Parser, w momencie napotkania takiego wywołania, przekazuje listę argumentów (argumenty w postaci wyrażeń są najpierw obliczane i przekazywana jest wartość liczbową) do metody *przypisz*, klasy *Funkcja*. Klasa ta zawiera drugą listę, z aktualnymi wartościami argumentów. Wspomniana metoda ustala wartości odpowiednich zmiennych i sprawdza poprawność wywołania – liczbę argumentów.

Po ustaleniu argumentów plik z funkcją zostaje przetworzony główną metodą kompilatora – *analiza*. Wynik działania (o ile funkcja coś zwraca) zostaje zapisany przez tę metodę tymczasowo do pliku `nazwa_funkcji.out`. Następnie:

- w przypadku funkcji użytej jako makro – wynik zostaje przepisany przez nadrzędnie działające metody, do pliku docelowego (lub pomocniczego),
- w przypadku funkcji używanej w wyrażeniach – zostaje zwrócony jako jeden z czynników danego wyrażenia i ewentualnie użyty w dalszych obliczeniach.

Na rysunku 4.7 przedstawiona została gramatyka deklaracji i wywołania funkcji.

$\langle \text{deklaracja funkcji} \rangle$	$\rightarrow$	<code><#function:<nazwa>{<nazwy argumentów>}
<kod wewnętrzny>
function#></code>
$\langle \text{nazwy argumentów} \rangle$	$::=$	<code>λ <nazwy></code>
$\langle \text{nazwy} \rangle$	$::=$	<code><zmienna> <nazwy>, <zmienna></code>
$\langle \text{wywołanie funkcji} \rangle$	$\rightarrow$	<code>_<nazwa>[<argumenty funkcji>]</code>
$\langle \text{argumenty funkcji} \rangle$	$::=$	<code>λ <argumenty></code>
$\langle \text{argumenty} \rangle$	$::=$	<code><wyrażenie> <argumenty>, <wyrażenie></code>
$\langle \text{nazwa} \rangle$	$\rightarrow$	<code>„ciąg liter i cyfr”</code>

Rysunek 4.7. Gramatyka deklaracji i wywołania funkcji

Źródło: opracowanie własne

4.2.7. Obsługa błędów

Podczas pracy nad programem K_T_EX, starano się zapewnić obsługę wszystkich błędów składni, jakie mogą wystąpić podczas użytkowania systemu. Autor wykorzystał do tego konstrukcje języka *Java* (wyjątki klas *String* i *IO*), jak i własne zabezpieczenia wbudowane w parser (wynikające z analizy składni języka K_T_EX). Komunikaty błędów składni, w wyrażeniach, mają postać np.:

```
!--- Linia: 23 - Błąd w~składni:  
"&a*3+a"  
  ^
```

Po operatorze "+" oczekiwano liczby, zmiennej lub funkcji.

Można tu było wskazać dokładną pozycję wystąpienia błędu, zapewniła to rekurencyjna budowa parsera – dzielenie wyrażeń na składowe (nieterminale).

Istnieją zabezpieczenia, dokonujące wstępnej analizy i kiedy wykryją błąd wyświetlony zostanie ogólny komunikat – bez wskazania pozycji w linii np.:

```
!--- Linia: 44 - Błąd w~wywołaniu funkcji "nazwa"!  
Zła ilość parametrów.
```

W niektórych przypadkach błędnego użycia systemu, niepowodujących jednak błędów wykonania kodu, wyświetlone zostaje ostrzeżenie np.:

```
!--- Linia: 225 - Ostrzeżenie:  
Wynik instrukcji: 1234567*10  
przekroczył zakres liczb LaTeX-a: 1.234567E7  
  ^
```

Może być wymagane ręczne poprawienie liczby w pliku wynikowym lub dokonanie zmian w algorytmie (skalowanie liczb do zakresu).

Komunikaty generowane przez kompilator K_T_EX:

- błędy dostępu do plików (błędy *IO*),
- błędy formatu liczby – użycie przecinka zamiast kropki jako separatora części ułamkowej,
- wywołanie niezdefiniowanej funkcji,
- odwołanie do niezadeklarowanej zmiennej,
- błędy wyrażeń arytmetycznych,
- brak końca deklaracji funkcji,
- dodatkowe komunikaty o tym, że błąd wystąpił:

- przy wywołaniu funkcji,
- po prawej stronie przypisania,
- podczas przetwarzania konkretnego pliku,
- ostrzeżenia (nie powodujące przerwania pracy systemu) o:
 - tekście występującym przy deklaracji funkcji, a który zostanie pominięty
 - otrzymaniu, w wyniku obliczeń, liczby w zapisie wykładniczym (nieobsługiwanym przez $\text{\LaTeX}$ -a).

4.3. Możliwości gotowego systemu

Pracując nad systemem $\text{K}\text{T}\text{E}\text{X}$, autor kierował się przede wszystkim jego poprawnym działaniem, oraz możliwościami rozwijania i szerokiego zastosowania systemu.

Program nie oferuje praktycznie żadnych wbudowanych komend generujących kod $\text{\LaTeX}$ -a. Istnieją narzędzia oferujące taką możliwość – używane do rysowania krzywych, wielokątów i innych kształtów (kody źródłowe dostępne są np. wraz z pracą [8]). $\text{K}\text{T}\text{E}\text{X}$ w to miejsce oferuje możliwość wykonywania osobiście zdefiniowanych wyrażeń i funkcji oraz implementację własnych algorytmów.

Przykłady praktycznych zastosowań $\text{K}\text{T}\text{E}\text{X}$ -a, w postaci kodu źródłowego, przedstawione są w załączniku B (od linii 189). W D.2 znajduje się końcowy efekt wykonania tego kodu.

Podsumowanie

L^AT_EX jest kompletnym narzędziem składu tekstów. System ten znalazł zastosowanie zarówno wśród profesjonalistów w składzie drukarskim, jak i wśród pracowników nauki, posługujących się językiem matematyki. L^AT_EX znalazł także uznanie wśród zwykłych użytkowników, którzy cenią sobie jakość i estetykę przygotowywanych dokumentów.

Jako system z założenia powstały w celu składu tekstów naukowych – z dużą ilością matematyki jak i treści informatycznych – posiada on jednak braki arytmetyczne. Odczuwalne są najbardziej przy programowaniu grafiki wektorowej, gdzie użytkownik musi wprowadzać szereg powtarzających się komend z argumentami liczbowymi, które muszą zostać najpierw przez niego obliczone.

W pracy przedstawiono koncepcję i realizację rozszerzenia L^AT_EX-a o środowisko programistyczne – preprocesor arytmetyczny. Autor, w pracy nad rozszerzeniem, nazwanym od swojego imienia K_TE_X, starał się stworzyć narzędzie, które pozwoli nie tylko w pełni wykorzystać możliwości L^AT_EX-a, ale też wprowadzić nową funkcjonalność. Ostatecznie powstał język programowania i kompilator, umożliwiające dołączanie do kodu źródłowego L^AT_EX-a własnych funkcji, makr i algorytmów wykonujących obliczenia na liczbach zmiennoprzecinkowych.

Dzięki systemowi K_TE_X możliwe stało się np. tworzenie tabel, w których liczona jest suma czy wartość średnia kolumny. Programując grafikę wektorową, można stworzyć funkcję tworzącą rysunek zależny od podanych argumentów wejściowych – pozwala to np. na skalowanie rysunków tylko w pionie lub tylko w poziomie (L^AT_EX pozwala na zmianę rozmiaru jednostek, co umożliwia skalowanie tylko z zachowaniem proporcji) czy dowolną zmianę proporcji. Natomiast tworzenie skomplikowanej grafiki wektorowej, jak np. wykresy funkcji, nie wymaga już korzystania z zewnętrznych programów, czy wykonywania obliczeń z użyciem kalkulatora i ręcznego wstawiania obliczonych wartości w definicji rysunku – wystarczy to zaprogramować.

Spis rysunków

1.1.	Polecenia kompilacji w $\text{\LaTeX}$ -u	
	<i>Źródło: opracowanie własne na podstawie [8, str. 1]</i>	7
3.1.	Plan pomieszczenia wykonany w $\text{\LaTeX}$ -u	
	<i>Źródło: opracowanie własne</i>	17
4.1.	Podstawowa gramatyka języka $\text{\TeX}$	
	<i>Źródło: opracowanie własne</i>	24
4.2.	Gramatyka tworzenia zmiennych tablicowych	
	<i>Źródło: opracowanie własne</i>	26
4.3.	Gramatyka wyrażeń języka $\text{\TeX}$	
	<i>Źródło: opracowanie własne</i>	27
4.4.	Gramatyka instrukcji <i>if</i>	
	<i>Źródło: opracowanie własne</i>	28
4.5.	Gramatyka wyrażeń logicznych	
	<i>Źródło: opracowanie własne</i>	30
4.6.	Gramatyka pętli <i>for</i>	
	<i>Źródło: opracowanie własne</i>	30
4.7.	Gramatyka deklaracji i wywołania funkcji	
	<i>Źródło: opracowanie własne</i>	32
D.1.	Plan pomieszczenia nr. 1	
	<i>Źródło: opracowanie własne</i>	63
D.2.	Plan pomieszczenia nr. 2	
	<i>Źródło: opracowanie własne</i>	63
D.3.	Plan pomieszczenia nr. 3	
	<i>Źródło: opracowanie własne</i>	63
D.4.	Układ współrzędnych z funkcją sinus	
	<i>Źródło: opracowanie własne</i>	64
D.5.	Rozwiązanie układu nierówności	
	<i>Źródło: opracowanie własne</i>	64

Spis tabel

1.1.	Pliki używane przez L ^A T _E X-a	
	<i>Źródło: opracowanie własne na podstawie [1, str. 4]</i>	7
D.1.	Przykład tabeli z zastosowaniem K _T E _X -a	
	<i>Dane na podstawie: Wikipedia – List of countries by carbon dioxide emissions</i> <i>http://en.wikipedia.org/wiki/List_of_countries_by_carbon_dioxide_emissions</i>	62

Bibliografia

- [1] Michel Goossens, Frank Mittelbach, Alexander Samarin, *The L^AT_EX Companion*, Addison Wesley Longman, Inc., 1994.
- [2] Hans Hagen, *ConT_EXt the manual*, 12 listopada 2001, [sieć, kwiecień 2009]
<http://www.pragma-ade.com/general/manuals/cont-eni.pdf>.
- [3] Donald E. Knuth, *T_EX and Metafont, New Directions in Typesetting*, The American Mathematical Society And Digital Press, Bedford, MA, 1979.
- [4] Leslie Lamport, *L^AT_EX: System opracowywania dokumentów*, Wydawnictwa Naukowo-Techniczne, Warszawa, 2004.
- [5] Paweł Mazur, *Jak szybko zacząć pisać w L^AT_EX?*, 21 Czerwca 2003, [sieć, kwiecień 2009] <http://www.ci.pwr.wroc.pl/~pmazur/LaTeX>.
- [6] Frank Mittelbach, Chris Rowley, *The L^AT_EX3 Project*, 12 Stycznia 1999, [sieć, marzec 2009] <http://www.latex-project.org/guides/ltx3info.pdf>.
- [7] Tobias Oetiker, *Nie za krótkie wprowadzenie do systemu L^AT_EX 2_ε*, Styczeń 2007, [sieć, kwiecień 2009] <http://www.ctan.org/tex-archive/info/lshort/polish/lshort2e.pdf>.
- [8] Urs Oswald, *Graphics in L^AT_EX 2_ε*, 10 Marca 2003, [sieć, marzec 2009]
<http://www.ursoswald.ch/LaTeXGraphics/overview/latexgraphics.pdf>.
- [9] Stefan Sokołowski, *Wykład przedmiotu „Języki formalne i metody kompilacji”*, semestr zimowy roku akademickiego 2008/2009, [sieć, maj 2009]
<http://student.pwsz.elblag.pl/~stefan/Dydaktyka/JezForm/Slajdy/>.

A. Gramatyka języka K_TE_X

$\langle \text{źródło} \rangle$	$\rightarrow$	λ
$\langle \text{źródło} \rangle$	$\rightarrow$	$\langle \text{źródło} \rangle \langle \text{treść} \rangle$
$\langle \text{źródło} \rangle$	$\rightarrow$	$\langle \text{źródło} \rangle \langle \text{komentarz} \rangle$
$\langle \text{źródło} \rangle$	$\rightarrow$	$\langle \text{źródło} \rangle \langle \text{kod} \rangle$
$\langle \text{treść} \rangle$	$\rightarrow$	„dowolny tekst z pominięciem par znaków: $\langle \# \ i \ \# \rangle$ ”
$\langle \text{komentarz} \rangle$	$\rightarrow$	// „dowolny tekst zakończony znakiem końca linii”
$\langle \text{kod} \rangle$	$\rightarrow$	$\langle \text{deklaracja funkcji} \rangle$
$\langle \text{kod} \rangle$	$\rightarrow$	$\langle \text{blok} \rangle$
$\langle \text{blok} \rangle$	$::=$	$\langle \text{for} \rangle \mid \langle \text{if} \rangle$
$\langle \text{blok} \rangle$	$\rightarrow$	$\langle \# \langle \text{instrukcje} \rangle \# \rangle$
$\langle \text{instrukcje} \rangle$	$::=$	$\lambda \mid \langle \text{instrukcje} \rangle \langle \text{instrukcja} \rangle ;$
$\langle \text{instrukcja} \rangle$	$\rightarrow$	$\langle \text{przypisanie} \rangle$
$\langle \text{instrukcja} \rangle$	$\rightarrow$	$\langle \text{in/de-krementacja} \rangle$
$\langle \text{instrukcja} \rangle$	$\rightarrow$	$\langle \text{wyrażenie} \rangle$
$\langle \text{przypisanie} \rangle$	$\rightarrow$	$\langle \text{zmienna} \rangle \langle \text{operator przypisania} \rangle \langle \text{wyrażenie} \rangle$
$\langle \text{operator przypisania} \rangle$	$::=$	$= \mid += \mid -= \mid *= \mid /=$
$\langle \text{in/de-krementacja} \rangle$	$::=$	$\langle \text{zmienna} \rangle ++ \mid \langle \text{zmienna} \rangle --$
$\langle \text{wyrażenie} \rangle$	$\rightarrow$	$\langle \text{wyrażenie} \rangle + \langle \text{składnik} \rangle$
$\langle \text{wyrażenie} \rangle$	$\rightarrow$	$\langle \text{wyrażenie} \rangle - \langle \text{składnik} \rangle$
$\langle \text{wyrażenie} \rangle$	$\rightarrow$	$\langle \text{składnik} \rangle$
$\langle \text{składnik} \rangle$	$\rightarrow$	$\langle \text{składnik} \rangle * \langle \text{czynnik} \rangle$
$\langle \text{składnik} \rangle$	$\rightarrow$	$\langle \text{składnik} \rangle / \langle \text{czynnik} \rangle$
$\langle \text{składnik} \rangle$	$\rightarrow$	$\langle \text{składnik} \rangle \% \langle \text{czynnik} \rangle$
$\langle \text{składnik} \rangle$	$\rightarrow$	$\langle \text{czynnik} \rangle$
$\langle \text{czynnik} \rangle$	$\rightarrow$	$\langle \text{atom} \rangle ^ \langle \text{czynnik} \rangle$
$\langle \text{czynnik} \rangle$	$\rightarrow$	$\langle \text{atom} \rangle$
$\langle \text{atom} \rangle$	$\rightarrow$	$(\langle \text{wyrażenie} \rangle)$
$\langle \text{atom} \rangle$	$\rightarrow$	$\langle \text{pobranie zmiennej} \rangle$
$\langle \text{atom} \rangle$	$\rightarrow$	$\langle \text{wywołanie funkcji} \rangle$
$\langle \text{atom} \rangle$	$\rightarrow$	$\langle \text{liczba} \rangle$
$\langle \text{pobranie zmiennej} \rangle$	$\rightarrow$	$\langle \text{zmienna} \rangle$
$\langle \text{liczba} \rangle$	$\rightarrow$	„ciąg cyfr (opcjonalnie z kropką i/lub minusem)”

$\langle \text{zmienna} \rangle$	$\rightarrow$	$\&\langle \text{nazwa} \rangle \langle \text{zmienna tab.} \rangle$
$\langle \text{zmienna tab.} \rangle$	$::=$	$\lambda \mid \langle \text{tablica} \rangle \mid \langle \text{zmienna tab.} \rangle \langle \text{tablica} \rangle \langle \text{nazwa} \rangle$
$\langle \text{tablica} \rangle$	$::=$	$[\langle \text{argumenty} \rangle] \mid \langle \text{tablica} \rangle [\langle \text{argumenty} \rangle]$
$\langle \text{if} \rangle$	$\rightarrow$	$\text{<\#if\{ \langle \text{wyrażenie log. *} \} } \\ \langle \text{kod wewnętrzny} \rangle \\ \text{if\#> } \langle \text{else} \rangle$
$\langle \text{else} \rangle$	$\rightarrow$	λ
$\langle \text{else} \rangle$	$\rightarrow$	$\text{<\#else } \\ \langle \text{kod wewnętrzny} \rangle \\ \text{else\#>}$
$\langle \text{wyrażenie log.} \rangle$	$\rightarrow$	$\langle \text{wyrażenie log.} \rangle \mid \mid \langle \text{składnik log.} \rangle$
$\langle \text{składnik log.} \rangle$	$\rightarrow$	$\langle \text{składnik log.} \rangle \&\& \langle \text{czynnik log.} \rangle$
$\langle \text{czynnik log.} \rangle$	$\rightarrow$	$\langle \text{logiczny} \rangle$
$\langle \text{logiczny} \rangle$	$\rightarrow$	$! \langle \text{logiczny} \rangle$
$\langle \text{logiczny} \rangle$	$\rightarrow$	$\langle \text{wyrażenie} \rangle \langle \text{operator porównania} \rangle \langle \text{wyrażenie} \rangle$
$\langle \text{operator porównania} \rangle$	$::=$	$== \mid => \mid = < \mid < > \mid > \mid <$
$\langle \text{for} \rangle$	$\rightarrow$	$\text{<\#for\{ \&\langle \text{nazwa} \rangle : \langle \text{wyrażenie} \rangle : \langle \text{wyrażenie} \rangle : \langle \text{wyrażenie} \rangle \} } \\ \langle \text{kod wewnętrzny} \rangle \\ \text{for\#>}$
$\langle \text{deklaracja funkcji} \rangle$	$\rightarrow$	$\text{<\#function: \langle \text{nazwa} \rangle \{ \langle \text{nazwy argumentów} \rangle \} } \\ \langle \text{kod wewnętrzny} \rangle \\ \text{function\#>}$
$\langle \text{nazwy argumentów} \rangle$	$::=$	$\lambda \mid \langle \text{nazwy} \rangle$
$\langle \text{nazwy} \rangle$	$::=$	$\langle \text{zmienna} \rangle \mid \langle \text{nazwy} \rangle, \langle \text{zmienna} \rangle$
$\langle \text{wywołanie funkcji} \rangle$	$\rightarrow$	$_ \langle \text{nazwa} \rangle [\langle \text{argumenty funkcji} \rangle]$
$\langle \text{argumenty funkcji} \rangle$	$::=$	$\lambda \mid \langle \text{argumenty} \rangle$
$\langle \text{argumenty} \rangle$	$::=$	$\langle \text{wyrażenie} \rangle \mid \langle \text{argumenty} \rangle, \langle \text{wyrażenie} \rangle$
$\langle \text{kod wewnętrzny} \rangle$	$\rightarrow$	λ
$\langle \text{kod wewnętrzny} \rangle$	$\rightarrow$	$\langle \text{kod wewnętrzny} \rangle \langle \text{treść} \rangle$
$\langle \text{kod wewnętrzny} \rangle$	$\rightarrow$	$\langle \text{kod wewnętrzny} \rangle \langle \text{blok} \rangle$
$\langle \text{nazwa} \rangle$	$\rightarrow$	$\text{„ciąg liter i cyfr”}$

B. Dokument prezentacyjny – kod źródłowy

```
1 %=====
2 % Dokument prezentacyjny dla systemu KTeX
3 % Kamil Piwowarczyk - V 2009
4 %=====
5 // Słowem wstępu dla oglądającego plik źródłowy KTeX-a:
6 // <-- Taki komentarz nie będzie widoczny w pliku wynikowym .tex
7 % Słowem wstępu dla oglądającego plik wynikowy KTeX-a:
8 % <-- Taki komentarz będzie widoczny w pliku wynikowym .tex
9 %=====
10
11 \paragraph{Uwaga:}
12 Dokument ten zaleca się czytać wraz z przeglądaniem jego kodu źródłowego, gdyż
13 to tam widoczne są wszystkie przykłady składni systemu KTeX.
14
15 \section{Przedstawienie składni systemu KTeX}
16
17 Działanie systemu opiera się na, „wtrąconych” w dokument LaTeX-a, blokach
18 instrukcji w języku KTeX. Bloki te wyróżnione są w pliku źródłowym KTeX-a
19 znakami: \texttt{<#} i \texttt{#>}. Instrukcje w takim bloku
20 muszą kończyć się znakiem \texttt{;} inaczej zostaną pominięte
21 (bez ostrzeżenia).
22 // Przykład bloku instrukcji:
23 <#Tą linię można potraktować jako komentarz -
24 ponieważ nie kończy się średnikiem zostanie pominięta.>
25
26 %-----
27 \subsubsection{Deklaracja zmiennych i działania na liczbach}
28
29 Identyfikatory zmiennych zaczynają się od znaku \texttt{\&}. Deklaracja
30 zmiennych w języku KTeX odbywa się niejawnie -- poprzez pierwsze przypisanie
31 im wartości.\\
32 W wyrażeniach arytmetycznych dostępne są operatory:
33 \begin{itemize}
34 \item przypisania: \cyt{=}, \cyt{+=}, \cyt{-=}, \cyt{*=}, \cyt{/=},
35 \item działań arytmetycznych: \cyt{+}, \cyt{-}, \cyt{*}, \cyt{/}, \cyt{\%},
36 \cyt{\textasciicircum},
37 \item inkrementacji: \cyt{++}, \cyt{--}.
38 \end{itemize}
39
40 W tym miejscu następuje zadeklarowanie dwóch zmiennych -- $a=1$ i $b=a+2=3$:
41 <# &a = 1;
42 &b = &a + 2; #> \mbox{[Koniec deklaracji]}
43 Deklaracja -- przypisanie jest „cichą” instrukcją -- nie daje widocznych
44 efektów w pliku wynikowym.
```

```

45
46 Działania na liczbach, nie użyte w przypisaniu, dają wynik w pliku wyjściowym.
47 Przykłady działań na zadeklarowanych powyżej zmiennych:
48 \begin{itemize}
49   \item  $a+b = \langle \#a+b; \# \rangle$ 
50   \item  $2b - (a+3a) = \langle \#2*b - (\&a+3*\&a); \# \rangle$ 
51   \item  $(-ba)/6 = \langle \#(-\&b*\&a)/6; \# \rangle$ 
52   \item  $b^{\frac{(b+2a)}{\text{mod} \frac{2b}{2a}}} = \langle \&b^{\frac{(\&b+2*\&a)}{((2*\&b)/(2*\&a))}}; \# \rangle$ 
53 \end{itemize}
54
55 Możliwe jest też deklarowanie zmiennych tablicowych. W K~TeX-u są to zmienne
56 zawierające w swojej nazwie liczbę w nawiasach: \cyt{[]}, \cyt{[]}.
57 Przykłady składni dla zmiennych tablicowych/indeksowanych:
58 \begin{itemize}
59   \item tab[7] (= 13)://
60    $\langle \&i=7; \&tab[7]=13; \# \rangle$   $\langle \&tab[\&i]; \# \rangle$ 
61   \item tab[7][9] (= tab[7]+7)://
62    $\langle \&j=2; \&tab[7][9]=\&tab[7]+\&i; \# \rangle$   $\langle \&tab[\&i][\&i+\&j]; \# \rangle$ 
63   \item tab[7,9] (= tab[7][9]+tab[7])://
64    $\langle \&tab[7,9]=\&tab[7][9]+\&tab[7]; \# \rangle$   $\langle \&tab[\&i,\&i+\&j]; \# \rangle$ 
65   \item zmienna[7]numerowana (= 23)://
66    $\langle \&zmienna[7]numerowana = 23; \# \rangle$   $\langle \&zmienna[7]numerowana; \# \rangle$ 
67 \end{itemize}
68
69 %-----
70 \subsubsection{Instrukcja warunkowa i operacje logiczne}
71
72 Składnia instrukcji warunkowej w K~TeX-u ma postać:\\
73 \texttt{<\#if\{\}warunek\texttt{\} \}\newline}
74 tekst i/lub instrukcje w blokach \newline
75 \texttt{if\#\>\}}
76 \texttt{<\#else\} ... \texttt{else\#\>\}}
77
78 W warunku, można stosować:
79 \begin{itemize}
80   \item operatory porównania: \cyt{==}, \cyt{>}, \cyt{<}, \cyt{<>},
81   \cyt{<>},
82   \item operatory logiczne: negacja -- \cyt{!}, alternatywa -- \cyt{||},
83   koniunkcja -- \cyt{\&\&}.
84 \end{itemize}
85
86 Instrukcje \emph{if} można dowolnie zagnieżdżać.\\
87 Przykład działania instrukcji warunkowych:
88 <\&ile=7;#>
89 \IF1:
90 <\#if{\&ile > 1 && \&ile < 13}//
91 Prawda: <\&ile;#> jest większe od 1 i mniejsze od 13
92 \IF2:
93 <\#if{!\&ile == 7}
94 Ta linia zostanie pominięta jeśli <\&ile;#> jest równe ,,7''.
95 if#> <\#else Else: <\&ile;#> jest równe ,,7'' else#>
96 if#>//
97
98 %-----
99 \subsubsection{Pętla \emph{for}}
```

```

100
101 W systemie K\TeX\ zaimplementowano pętlę \emph{for}. Definiuje się ją
102 specjalnym blokiem:
103 \texttt{<\#for\{\&zmienna\_sterująca:}od\texttt{:}krok\texttt{:}do\texttt{\}}
104 \newline
105 tekst i/lub instrukcje w~blokach \newline
106 \texttt{for\#{>}}
107
108 Jak widać powyżej, przyjęto deklarację z~podaniem nazwy zmiennej sterującej
109 i~jej zakresu wraz z~krokiem inkrementacji (lub dekrementacji).
110 Parametry \emph{od}, \emph{krok} i~\emph{do} są, tak jak wszystkie zmienne
111 w K\TeX, typu float. W ich miejscu można stosować także wyrażenia i~wywołania
112 funkcji. Pętla można zagnieźdźać oraz wpływać wewnątrz nich na zmienne
113 sterujące.\\
114 Przykład efektu działania trzech zagnieźdzonych pętli:\\
115 <\#for{&i:0:0.1:10}//ta pętla zostanie naprawdę wykonana tylko raz...
116 <\#for{&j:4:-2:0}//ta pętla wpływa na ilość obrotów następnej pętli
117 <\#for{&k:0:1:&j/2}//wewnątrz tej pętli zmiana ulega zmienna "i":
118 <\#&i+=2;#>
119 (i, j, k) = (<\#&i;#>, <\#&j;#>, <\#&k;#>)\
120 for#>
121 for#>
122 for#>
123
124 Pętla \emph{for} działająca jak pętla \emph{do/while} (niezależna od pętli
125 zmienna jest zwiększana dopóki jest mniejsza od 10):
126 <\#&z=0;#>
127 <\#for{&i:0:0:1}
128 <\#&z++;&z;#>, //
129 <\#if{&z => 10}Przerwanie!<\#&i=2;#>if#>// while z<10
130 for#>
131
132 %-----
133 \subsubsection{Deklaracja i~użycie funkcji}
134
135 Deklaracji funkcji dokonuje się za pomocą specjalnego bloku:\\
136 \texttt{<\#function:nazwa\{\&arg1 , \&arg2, ...\} } \newline
137 kod funkcji \newline
138 \texttt{function\#{>}}
139
140 Funkcji nie można deklarować wewnątrz pętli ani instrukcji warunkowych.
141 Wewnątrz bloku funkcji może znajdować się zwykły tekst (zostanie przepisany do
142 pliku wynikowego w~momencie wywołania funkcji) oraz -- przede wszystkim --
143 bloki instrukcji.\\
144 Wywołanie odbywa się poprzez podanie nazwy poprzedzonej znakiem ,,\_''
145 i~argumentów w~nawiasach: \cyt{[ i~\cyt{]}, oddzielonych przecinkami -- np.:
146 \ \texttt{\_nazwa[arg1, arg2]}.
147
148 Standardowe użycie funkcji wymaga, aby w~kodzie nie wypisywano tekstu (spacje
149 też są liczone jako tekst, ale tabulatory nie -- są automatycznie pomijane),
150 a~wypisana została tylko jedna liczba -- wynik funkcji.
151
152 W tym miejscu zostanie zadeklarowana funkcja \emph{dodaj}, która dodaje do
153 siebie dwa pobierane argumenty i~podaje wynik:
154 <\#function:dodaj{\&arg1,&arg2}

```

```

155      <#&arg1+&arg2;#> //W treści funkcji nie mogą występować spacje!
156 function#>
157 \mbox{[koniec]}
158
159 \noindent Tutaj wywołanie funkcji \emph{dodaj} dla obu argumentów równych 2:
160 <#_dodaj[2,2];#>.
161
162 \noindent Taką funkcję można używać w~wyrażeniach arytmetycznych np.
163 \texttt{2\textasciicircum(\_dodaj[1,2])}: <#2^(\_dodaj[1,2]);#>.
164
165 Inne użycia (nie mogą występować w~wyrażeniach):
166 \begin{itemize}
167   \item jako procedura -- wykonuje obliczenia, zmienia wartości zmiennych
168   (poprzez przypisanie), ale sama nic nie wypisuje.\\
169   Przykład:\\
170   Zadeklarowanie funkcji bezargumentowej ustalającej zmienne $a=7$ i~$b=13$:
171     <#function:siedem{}
172       <#&a=7; &b=13;#>
173       function#>
174     \mbox{[koniec]}\\
175     Wywołanie: <#_siedem[];#> \mbox{[koniec]}. Wypisanie zmiennych $a$ i~$b$:
176     <#&a;#> <#&b;#>.
177
178   \item jako makro -- mogące wypisywać tekst, wartości zmiennych itp.\\
179   Przykład:\\
180   Zadeklarowanie funkcji pobierającej liczbę i~wypisującej ją wraz z~tekstem:
181     <#function:liczba{&ile}
182       Podana liczba to: <#&ile;#> %zdecydowanie za mało...
183       function#>
184     \mbox{[koniec]}\\
185     Wywołanie: <#_liczba[100];#>
186 \end{itemize}
187
188 %=====
189 \section{Praktyczne przykłady zastosowań K\TeX-a}\label{praktyczne}
190
191 %-----
192 \subsubsection{Tabela z~sumą i~średnią wartością kolumny}
193
194 K\TeX\ pozwala również na wykonywanie obliczeń i~wykorzystanie ich wyników
195 jako zwykły tekst. Poniżej pokazano przykład tabeli, w~której policzona
196 została suma i~średnia wartość kolumny. Zmieniając jedną z~wartości w~pliku
197 źródłowym wynik zostanie zaktualizowany.
198
199 \begin{table}[h]
200   \centering
201
202   \caption{Przykład tabeli z~zastosowaniem K\TeX-a \protect \\
203   \emph{Dane na podstawie:
204   Wikipedia -- List of countries by carbon dioxide emissions} \protect \\
205   \emph{http://<##>/en.wikipedia.org/wiki/List\_of\_countries\_by\_carbon%
206   \_dioxide\_emissions}}
207   % Początek deklaracji tabeli:
208   \footnotesize{\begin{tabular}{|r|l|r|} \hline \hline
209

```

```

210 Pozycja      & Państwo      & Roczna emisja $CO_2$      \\
211 na świecie & & w~milionach ton      \\ \hline\hline
212 1            & USA            & <#&em[1]=6049;&em[1];#> \\ \hline
213 2            & Chiny i~Tajwan & <#&em[2]=5010;&em[2];#> \\ \hline
214 3            & Rosja          & <#&em[3]=1524;&em[3];#> \\ \hline
215 4            & Indie          & <#&em[4]=1342;&em[4];#> \\ \hline
216 5            & Japonia        & <#&em[5]=1257;&em[5];#> \\ \hline
217 6            & Niemcy         & <#&em[6]=860;&em[6];#> \\ \hline
218 7            & Kanada         & <#&em[7]=639;&em[7];#> \\ \hline
219 8            & W. Brytania    & <#&em[8]=587;&em[8];#> \\ \hline
220 9            & Płn. Korea    & <#&em[9]=465;&em[9];#> \\ \hline
221 10           & Włochy        & <#&em[10]=449;&em[10];#> \\ \hline
222 \multicolumn{3}{|c|}{...}\hline
223 21            & Polska        & <#&em[11]=307;&em[11];#> \\ \hline\hline
224 //Suma i~średnia:
225 \multicolumn{1}{c}{ } & \multicolumn{1}{|r|}{Suma:} & //
226 <#&suma=0;#><#for{&i:1:1:11}<#&suma+=&em[&i];#>for#><#&suma;#> \\ \cline{2-3}
227 \multicolumn{1}{c}{ } & \multicolumn{1}{|r|}{Średnia:} & //
228 <#&suma/11;#>
229 \\ \cline{2-3}\cline{2-3}
230 \end{tabular}
231 \end{table}
232
233 %-----
234 \subsubsection{Plan pomieszczenia, ze zmianą proporcji}
235
236 W~rozdziale \ref{braki} przedstawiono prosty rysunek pokoju (\ref{plan1})
237 i~wykazano trudność w~jego aktualizacji -- zmianie położenia drzwi, czy
238 zmniejszenie grubości ścian.
239 Poniżej znajduje się deklaracja funkcji K\TeX-a, która rysuje ten plan zgodnie
240 z~podanymi parametrami: szerokość, długość, pozycja drzwi, szerokość drzwi
241 i~grubość ściany. Następnie wywołano ją trzy razy z~różnymi argumentami.
242
243 <#function:pokoj{&szer,&dlug,&a,&b,&g}
244 \begin{figure}[h]
245   \centering
246   \setlength{\unitlength}{0.5mm}
247
248   \begin{picture}(<#&szer;#>,<#&dlug;#>)
249     \put(0,<#&dlug;#>){\line(1,0){<#&a;#>}}
250     \put(<#&g;#>,<#&dlug-&g;#>){\line(1,0){<#&a-&g;#>}}
251     \put(<#&a;#>,<#&dlug-&g;#>){\line(0,1){<#&g;#>}}
252     \put(<#&a+&b;#>,<#&dlug;#>){\line(1,0){<#&szer-&a-&b;#>}}
253     \put(<#&a+&b;#>,<#&dlug-&g;#>){\line(1,0){<#&szer-&a-&b-&g;#>}}
254     \put(<#&a+&b;#>,<#&dlug-&g;#>){\line(0,1){<#&g;#>}}
255     \put(0,0){\line(0,1){<#&dlug;#>}}
256     \put(<#&g;#>,<#&g;#>){\line(0,1){<#&dlug-2*&g;#>}}
257     \put(<#&szer;#>,0){\line(0,1){<#&dlug;#>}}
258     \put(<#&szer-&g;#>,<#&g;#>){\line(0,1){<#&dlug-2*&g;#>}}
259     \put(0,0){\line(1,0){<#&szer;#>}}
260     \put(<#&g;#>,<#&g;#>){\line(1,0){<#&szer-2*&g;#>}}
261   \end{picture}
262
263   \caption{Plan pomieszczenia nr. <#&ktory;#> \protect \\
264   \emph{Źródło: opracowanie własne}}

```

```

265 \end{figure}
266 <#&ktory++;#>
267 function#>
268 <#&ktory=1;#>
269
270 %Pokój 1
271 <#_pokoj[80,50,20,20,5];#>
272
273 %Pokój 2
274 <#_pokoj[40,60,20,10,3];#>
275
276 %Pokój 3
277 <#_pokoj[100,30,15,5,10];#>
278
279 %-----
280 \subsubsection{Układ współrzędnych z funkcją sinus}
281
282 W tym przykładzie zastosowano K\TeX-a do wypisania jednostek na osiach oraz do
283 wygenerowania ciągu przybliżonych wartości sinusa (z szeregu Taylora) -- za
284 pomocą zaprogramowanych funkcji, które posłużyły do narysowania wykresu. Wynik
285 działania obrazuje rysunek \ref{sinus}.
286
287 % Tutaj zadeklarowano funkcje
288 // Kod funkcji silnia:
289 <#function:silnia{&n}
290 <#if{&n==1}
291 <#1;#>
292 if#> <#else
293 <#&n*_silnia[&n-1];#>
294 else#>
295 function#>
296 //
297 // Kod funkcji przybliżającej wartość sinusa z szeregu Taylora:
298 <#function:sin{&x}
299 <#&wyn=&x;&znak=1;#>
300 <#for{&k:3:2:20}
301 <#&znak*=-1;#>
302 <#&wyn+=(&x^&k/_silnia[&k])*&znak;#>
303 for#>
304 <#&wyn;#>
305 function#>
306 //
307 // Kod generujący rysunek:
308 % Gotowy kod generujący rysunek:
309 \begin{figure}[h]
310 \centering
311 \setlength{\unitlength}{1cm}
312
313 \begin{pspicture}(15,5)
314
315 % Jednostki na osi y:
316 \put(0,2.5){\vector(1,0){15}}
317 <#for{&i:-2:0.5:2}
318 <#if{&i<>0}//
319 \put(7.4,<#2.5+&i;#>){\line(1,0){0.2} \put(0.1,-0.07){\scriptsize{<#&i;#>}}}

```

```

320 if#>//
321 for#>
322 \put(7.6,<#3+&i;#>){$y$}
323
324 % Jednostki na osi x:
325 \put(7.5,0){\vector(0,1){5}}
326 <#for{&i:-2:0.5:2}
327 <#if{&i<>0}//
328 \put(<#7.5+&i*3.14;#>,2.4){\line(0,1){0.2} //
329 \put(-0.2,0.25){\scriptsize{<#&i;#>$\pi$}}}
330 if#>//
331 for#>
332 \put(<#8.5+&i*3.14;#>,2.6){$x$}
333 \put(7.2,2.65){\scriptsize{0}}
334
335 % Rysowanie sinusa:
336 <#&t1i=7.5-2*3.14;&t1s=2.5;#>
337 <#for{&i:-2:0.1:2}
338 <#&t2i=7.5+&i*3.14; &t2s=2.5+_sin[&i*3.14];#>
339 \psline(<#&t1i;#>,<#&t1s;#>)(<#&t2i;#>,<#&t2s;#>)
340 <#&t1i=&t2i;&t1s=&t2s;#>
341 for#>
342
343 \end{pspicture}
344 \linespread{1}
345 \caption{Układ współrzędnych z~funkcją sinus \protect \\
346 \emph{Źródło: opracowanie własne}}
347 \label{sinus}
348 \end{figure}
349
350 %-----
351 \subsubsection{Graficzne rozwiązanie układu nierówności}
352
353 Następny przykład zawiera zakreślony obszar,
354 będący rozwiązaniem układu nierówności:
355 $$\left\{
356 \begin{array}{c}
357 \frac{x^2}{5} < y \\
358 -x^2 + 2x + 2 > y
359 \end{array} \right. $$
360
361 % Kod generujący rysunek:
362 \begin{figure}[h]
363 \centering
364 \psset{unit=0.8cm}
365 \begin{pspicture}(10,11)
366
367 % Jednostki na osi y:
368 \put(5,0){\vector(0,1){11}}
369 <#for{&i:-5:1:5}
370 <#if{&i<>0}//
371 \put(4.9,<#5+&i;#>){\line(1,0){0.2} \put(0.1,-0.07){\scriptsize{<#&i;#>}}}
372 if#>//
373 for#>
374 \put(5.1,<#6+&i;#>){$y$}

```

```

375
376 % Jednostki na osi x:
377 \put(0,5){\vector(1,0){11}}
378 <#for{&i:-5:1:5}
379 <#if{&i<>0}//
380 \put(<#5+&i;#>,4.9){\line(0,1){0.2} \put(-0.1,0.25){\scriptsize{<#&i;#>}}}
381 if#>//
382 for#>
383 \put(<#6+&i;#>,5.1){$x$}
384 \put(5.2,5.15){\scriptsize{0}}
385
386 % Rysowanie pierwszej paraboli - \ / :
387 <#&x1=-4;&y1=&x1^2/5;#>
388 <#for{&i:-4:0.2:4}
389 <#&x2=&i;&y2=&x2^2/5;#>
390 \psline(<#&x1+5;#>,<#&y1+5;#>)(<#&x2+5;#>,<#&y2+5;#>)
391 <#&x1=&x2;&y1=&y2;#>
392 for#>
393
394 % Rysowanie drugiej paraboli - /\ :
395 <#&x1=-1.6;&y1=-1*&x1^2 + 2*&x1 + 2;#>
396 <#for{&i:-1.6:0.2:3.6}
397 <#&x2=&i; &y2=-1*&x2^2 + 2*&x2 + 2;#>
398 \psline(<#&x1+5;#>,<#&y1+5;#>)(<#&x2+5;#>,<#&y2+5;#>)
399 <#&x1=&x2;&y1=&y2;#>
400 for#>
401
402 % Zakreskowanie rozwiązania:
403 \psset{linewidth=.01mm}
404 <#for{&i:-1.6:0.1:3.6}
405 <#&x1=&i;&y1=&x1^2/5;#>
406 <#&x2=&i;&y2=-1*&x2^2 + 2*&x2 + 2;#>
407 //<#&x3=&i-0.4;&y3=-2*&x3+2*&i;#>
408 //<#&x4=&i; &y4=-2*&x4+2*&i;#>
409 <#if{&y2=>&y1}//
410 \psline(<#&x1+5;#>,<#&y1+5;#>)(<#&x2+5;#>,<#&y2+5;#>)
411 if#>//
412 for#>
413
414 \end{pspicture}
415 \linespread{1}
416 \caption{Rozwiązanie układu nierówności \protect \\
417 \emph{Źródło: opracowanie własne}}
418 \end{figure}

```

C. Dokument prezentacyjny – kod po przetworzeniu K_TE_X-em

```
1 %=====
2 % Dokument prezentacyjny dla systemu KTeX
3 % Kamil Piwowarczyk - V 2009
4 %=====
5 % Słowem wstępu dla oglądającego plik wynikowy KTeX-a:
6 % <-- Taki komentarz będzie widoczny w pliku wynikowym .tex
7 %=====
8
9 \paragraph{Uwaga:}
10 Dokument ten zaleca się czytać wraz z przeglądaniem jego kodu źródłowego, gdyż
11 to tam widoczne są wszystkie przykłady składni systemu KTeX.
12
13 \section{Przedstawienie składni systemu KTeX}
14
15 Działanie systemu opiera się na, „wtrąconych” w dokument LaTeX-a, blokach
16 instrukcji w języku KTeX. Bloki te wyróżnione są w pliku źródłowym KTeX-a
17 znakami: \texttt{<#} i \texttt{#>}. Instrukcje w takim bloku
18 muszą kończyć się znakiem \texttt{;} inaczej zostaną pominięte
19 (bez ostrzeżenia).
20
21
22 %-----
23 \subsubsection{Deklaracja zmiennych i działania na liczbach}
24
25 Identyfikatory zmiennych zaczynają się od znaku \texttt{\&}. Deklaracja
26 zmiennych w języku KTeX odbywa się niejawnie -- poprzez pierwsze przypisanie
27 im wartości.\\
28 W wyrażeniach arytmetycznych dostępne są operatory:
29 \begin{itemize}
30 \item przypisania: \cyt{=}, \cyt{+=}, \cyt{-=}, \cyt{*=}, \cyt{/=},
31 \item działań arytmetycznych: \cyt{+}, \cyt{-}, \cyt{*}, \cyt{/}, \cyt{\%},
32 \cyt{\textasciicircum},
33 \item inkrementacji: \cyt{++}, \cyt{--}.
34 \end{itemize}
35
36 W tym miejscu następuje zadeklarowanie dwóch zmiennych -- $a=1$ i $b=a+2=3$:
37 \mbox{[Koniec deklaracji]}
38 Deklaracja -- przypisanie jest „cichą” instrukcją -- nie daje widocznych
39 efektów w pliku wynikowym.
40
41 Działania na liczbach, nie użyte w przypisaniu, dają wynik w pliku wyjściowym.
42 Przykłady działań na zadeklarowanych powyżej zmiennych:
43 \begin{itemize}
44 \item $a+b=4$
```

```

45 \item  $2b-(a+3a)=2\$$ 
46 \item  $\$(-ba)/6=-0.5\$$ 
47 \item  $\$b^{\{((b+2a)\ \bmod\ \frac{2b}{2a})\}}=9\$$ 
48 \end{itemize}
49
50 Możliwe jest też deklarowanie zmiennych tablicowych. W K\TeX-u są to zmienne
51 zawierające w swojej nazwie liczbę w nawiasach: \cyt{[]}, \cyt{[]}.
52 Przykłady składni dla zmiennych tablicowych/indeksowanych:
53 \begin{itemize}
54 \item tab[7] (= 13): 13
55 \item tab[7][9] (=tab[7]+7): 20
56 \item tab[7,9] (= tab[7][9]+tab[7]): 33
57 \item zmienna[7]numerowana (= 23): 23
58 \end{itemize}
59
60 %-----
61 \subsubsection{Instrukcja warunkowa i operacje logiczne}
62
63 Składnia instrukcji warunkowej w K\TeX-u ma postać:\\
64 \texttt{<\#if\{\}warunek\texttt{\} \}\newline}
65 tekst i/lub instrukcje w blokach \newline
66 \texttt{if\#\{>}}
67 \texttt{<\#else\} ... \texttt{else\#\{>}}
68
69 W warunku, można stosować:
70 \begin{itemize}
71 \item operatory porównania: \cyt{==}, \cyt{>}, \cyt{<}, \cyt{<}, \cyt{>},
72 \cyt{<>},
73 \item operatory logiczne: negacja -- \cyt{!}, alternatywa -- \cyt{||},
74 koniunkcja -- \cyt{\&\&}.
75 \end{itemize}
76
77 Instrukcje \emph{if} można dowolnie zagnieżdżać.\\
78 Przykład działania instrukcji warunkowych:
79 \\IF1:
80 Prawda: 7 jest większe od 1 i mniejsze od 13
81 \\IF2:
82 Else: 7 jest równe ,,7''
83
84 %-----
85 \subsubsection{Pętla \emph{for}}
86
87 W systemie K\TeX\ zaimplementowano pętlę \emph{for}. Definiuje się ją
88 specjalnym blokiem:
89 \texttt{<\#for\{\&zmienna\_sterująca\}od\texttt{:}krok\texttt{:}do\texttt{\}}\}
90 \newline
91 tekst i/lub instrukcje w blokach \newline
92 \texttt{for\#\{>}}
93
94 Jak widać powyżej, przyjęto deklarację z podaniem nazwy zmiennej sterującej
95 i jej zakresu wraz z krokiem inkrementacji (lub dekrementacji).
96 Parametry \emph{od}, \emph{krok} i \emph{do} są, tak jak wszystkie zmienne
97 w K\TeX, typu float. W ich miejscu można stosować także wyrażenia i wywołania
98 funkcji. Pętle można zagnieżdżać oraz wpływać wewnątrz nich na zmienne
99 sterujące.\\

```

```

100 Przykład efektu działania trzech zagnieżdżonych pętli:\\
101 (i, j, k) = (2, 4, 0)\\
102 (i, j, k) = (4, 4, 1)\\
103 (i, j, k) = (6, 4, 2)\\
104 (i, j, k) = (8, 2, 0)\\
105 (i, j, k) = (10, 2, 1)\\
106 (i, j, k) = (12, 0, 0)\\
107
108 Pętla \emph{for} działająca jak pętla \emph{do/while} (niezależna od pętli
109 zmienna jest zwiększana dopóki jest mniejsza od 10):
110 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, Przerwanie!
111
112
113 %-----
114 \subsubsection{Deklaracja i~użycie funkcji}
115
116 Deklaracji funkcji dokonuje się za pomocą specjalnego bloku:\\
117 \texttt{<#function:nazwa\{&arg1 , &arg2, ...& \} \newline
118 kod funkcji \newline
119 \texttt{function\#{>}}
120
121 Funkcji nie można deklarować wewnątrz pętli ani instrukcji warunkowych.
122 Wewnątrz bloku funkcji może znajdować się zwykły tekst (zostanie przepisany do
123 pliku wynikowego w~momencie wywołania funkcji) oraz -- przede wszystkim --
124 bloki instrukcji.\\
125 Wywołanie odbywa się poprzez podanie nazwy poprzedzonej znakiem ,,\_’’
126 i~argumentów w~nawiasach: \cyt{[] i~\cyt{[]}, oddzielonych przecinkami -- np.:
127 \\ \texttt{\_nazwa[arg1, arg2]}.
128
129 Standardowe użycie funkcji wymaga, aby w~kodzie nie wypisywano tekstu (spacje
130 też są liczone jako tekst, ale tabulatory nie -- są automatycznie pomijane),
131 a~wypisana została tylko jedna liczba -- wynik funkcji.
132
133 W tym miejscu zostanie zadeklarowana funkcja \emph{dodaj}, która dodaje do
134 siebie dwa pobierane argumenty i~podaje wynik:
135 \mbox{[koniec]}
136
137 \noindent Tutaj wywołanie funkcji \emph{dodaj} dla obu argumentów równych 2:
138 4.
139
140 \noindent Taką funkcję można używać w~wyrażeniach arytmetycznych np.
141 \texttt{2\textasciicircum(\_dodaj[1,2])}: 8.
142
143 Inne użycia (nie mogą występować w~wyrażeniach):
144 \begin{itemize}
145   \item jako procedura -- wykonuje obliczenia, zmienia wartości zmiennych
146     (poprzez przypisanie), ale sama nic nie wypisuje.\\
147   Przykład:\\
148   Zadeklarowanie funkcji bezargumentowej ustalającej zmienne $a=7$ i~$b=13$:
149   \mbox{[koniec]}\\
150   Wywołanie: \mbox{[koniec]}. Wypisanie zmiennych $a$ i~$b$:
151   7 13.
152
153   \item jako makro -- mogące wypisywać tekst, wartości zmiennych itp.\\
154   Przykład:\\

```

```

155   Zadeklarowanie funkcji pobierającej liczbę i~wypisującej ją wraz z~tekstem:
156   \mbox{[koniec]}\
157   Wywołanie: Podana liczba to: 100 %zdecydowanie za mało...
158   \end{itemize}
159
160   %=====
161   \section{Praktyczne przykłady zastosowań K\TeX-a}\label{praktyczne}
162
163   %-----
164   \subsubsection{Tabelka z~sumą i~średnią wartością kolumny}
165
166   K\TeX\ pozwala również na wykonywanie obliczeń i~wykorzystanie ich wyników
167   jako zwykły tekst. Poniżej pokazano przykład tabeli, w~której policzona
168   została suma i~średnia wartość kolumny. Zmieniając jedną z~wartości w~pliku
169   źródłowym wynik zostanie zaktualizowany.
170
171   \begin{table}[h]
172     \centering
173
174     \caption{Przykład tabeli z~zastosowaniem K\TeX-a \protect \\
175     \emph{Dane na podstawie:
176     Wikipedia -- List of countries by carbon dioxide emissions} \protect \\
177     \emph{http://en.wikipedia.org/wiki/List\_of\_countries\_by\_carbon%
178     \_dioxide\_emissions}}
179     % Początek deklaracji tabeli:
180     \footnotesize{\begin{tabular}{|r|l|r|} \hline\hline
181
182     Pozycja      & Państwo      & Roczna emisja $CO_2$      & \\
183     na świecie & & & \\
184     1            & & & \\
185     2            & & & \\
186     3            & & & \\
187     4            & & & \\
188     5            & & & \\
189     6            & & & \\
190     7            & & & \\
191     8            & & & \\
192     9            & & & \\
193     10           & & & \\
194     \multicolumn{3}{|c|}{...}\hline
195     21           & & & \\
196     \multicolumn{1}{c}{} & \multicolumn{1}{|r|}{Suma:} & 18489 & \\
197     \multicolumn{1}{c}{} & \multicolumn{1}{|r|}{Średnia:} & 1680.8182 & \\
198     \ccline{2-3}\cline{2-3}
199     \end{tabular}}
200   \end{table}
201
202   %-----
203   \subsubsection{Plan pomieszczenia, ze zmianą proporcji}
204
205   W~rozdziale \ref{braki} przedstawiono prosty rysunek pokoju (\ref{plan1})
206   i~wykazano trudność w~jego aktualizacji -- zmianie położenia drzwi, czy
207   zmniejszenie grubości ścian.
208   Poniżej znajduje się deklaracja funkcji K\TeX-a, która rysuje ten plan zgodnie
209   z~podanymi parametrami: szerokość, długość, pozycja drzwi, szerokość drzwi

```

```

210 i~grubość ściany. Następnie wywołano ją trzy razy z~różnymi argumentami.
211
212
213 %Pokój 1
214 \begin{figure}[h]
215   \centering
216   \setlength{\unitlength}{0.5mm}
217
218   \begin{picture}(80,50)
219     \put(0,50){\line(1,0){20}}
220     \put(5,45){\line(1,0){15}}
221     \put(20,45){\line(0,1){5}}
222     \put(40,50){\line(1,0){40}}
223     \put(40,45){\line(1,0){35}}
224     \put(40,45){\line(0,1){5}}
225     \put(0,0){\line(0,1){50}}
226     \put(5,5){\line(0,1){40}}
227     \put(80,0){\line(0,1){50}}
228     \put(75,5){\line(0,1){40}}
229     \put(0,0){\line(1,0){80}}
230     \put(5,5){\line(1,0){70}}
231   \end{picture}
232
233   \caption{Plan pomieszczenia nr. 1 \protect \\\
234     \emph{Źródło: opracowanie własne}}
235 \end{figure}
236 %Pokój 2
237 \begin{figure}[h]
238   \centering
239   \setlength{\unitlength}{0.5mm}
240
241   \begin{picture}(40,60)
242     \put(0,60){\line(1,0){20}}
243     \put(3,57){\line(1,0){17}}
244     \put(20,57){\line(0,1){3}}
245     \put(30,60){\line(1,0){10}}
246     \put(30,57){\line(1,0){7}}
247     \put(30,57){\line(0,1){3}}
248     \put(0,0){\line(0,1){60}}
249     \put(3,3){\line(0,1){54}}
250     \put(40,0){\line(0,1){60}}
251     \put(37,3){\line(0,1){54}}
252     \put(0,0){\line(1,0){40}}
253     \put(3,3){\line(1,0){34}}
254   \end{picture}
255
256   \caption{Plan pomieszczenia nr. 2 \protect \\\
257     \emph{Źródło: opracowanie własne}}
258 \end{figure}
259 %Pokój 3
260 \begin{figure}[h]
261   \centering
262   \setlength{\unitlength}{0.5mm}
263
264   \begin{picture}(100,30)

```

```

265 \put(0,30){\line(1,0){15}}
266 \put(10,20){\line(1,0){5}}
267 \put(15,20){\line(0,1){10}}
268 \put(20,30){\line(1,0){80}}
269 \put(20,20){\line(1,0){70}}
270 \put(20,20){\line(0,1){10}}
271 \put(0,0){\line(0,1){30}}
272 \put(10,10){\line(0,1){10}}
273 \put(100,0){\line(0,1){30}}
274 \put(90,10){\line(0,1){10}}
275 \put(0,0){\line(1,0){100}}
276 \put(10,10){\line(1,0){80}}
277 \end{picture}
278
279 \caption{Plan pomieszczenia nr. 3 \protect \\
280 \emph{Źródło: opracowanie własne}}
281 \end{figure}
282 %-----
283 \subsubsection{Układ współrzędnych z funkcją sinus}
284
285 W tym przykładzie zastosowano KTTeX-a do wypisania jednostek na osiach oraz do
286 wygenerowania ciągu przybliżonych wartości sinusa (z szeregu Taylora) -- za
287 pomocą zaprogramowanych funkcji, które posłużyły do narysowania wykresu. Wynik
288 działania obrazuje rysunek \ref{sinus}.
289
290 % Tutaj zadeklarowano funkcje
291 % Gotowy kod generujący rysunek:
292 \begin{figure}[h]
293 \centering
294 \setlength{\unitlength}{1cm}
295
296 \begin{pspicture}(15,5)
297
298 % Jednostki na osi y:
299 \put(0,2.5){\vector(1,0){15}}
300 \put(7.4,0.5){\line(1,0){0.2} \put(0.1,-0.07){\scriptsize{-2}}}
301 \put(7.4,1){\line(1,0){0.2} \put(0.1,-0.07){\scriptsize{-1.5}}}
302 \put(7.4,1.5){\line(1,0){0.2} \put(0.1,-0.07){\scriptsize{-1}}}
303 \put(7.4,2){\line(1,0){0.2} \put(0.1,-0.07){\scriptsize{-0.5}}}
304 \put(7.4,3){\line(1,0){0.2} \put(0.1,-0.07){\scriptsize{0.5}}}
305 \put(7.4,3.5){\line(1,0){0.2} \put(0.1,-0.07){\scriptsize{1}}}
306 \put(7.4,4){\line(1,0){0.2} \put(0.1,-0.07){\scriptsize{1.5}}}
307 \put(7.4,4.5){\line(1,0){0.2} \put(0.1,-0.07){\scriptsize{2}}}
308
309 \put(7.6,5){\mathstrut}
310
311 % Jednostki na osi x:
312 \put(7.5,0){\vector(0,1){5}}
313 \put(1.22,2.4){\line(0,1){0.2} \put(-0.2,0.25){\scriptsize{-2$\pi$}}}
314 \put(2.79,2.4){\line(0,1){0.2} \put(-0.2,0.25){\scriptsize{-1.5$\pi$}}}
315 \put(4.36,2.4){\line(0,1){0.2} \put(-0.2,0.25){\scriptsize{-1$\pi$}}}
316 \put(5.93,2.4){\line(0,1){0.2} \put(-0.2,0.25){\scriptsize{-0.5$\pi$}}}
317 \put(9.07,2.4){\line(0,1){0.2} \put(-0.2,0.25){\scriptsize{0.5$\pi$}}}
318 \put(10.64,2.4){\line(0,1){0.2} \put(-0.2,0.25){\scriptsize{1$\pi$}}}
319 \put(12.21,2.4){\line(0,1){0.2} \put(-0.2,0.25){\scriptsize{1.5$\pi$}}}

```

```

320 \put(13.78,2.4){\line(0,1){0.2} \put(-0.2,0.25){\scriptsize{2$\pi$}}}
321
322 \put(14.78,2.6){$x$}
323 \put(7.2,2.65){\scriptsize{0}}
324
325 % Rysowanie sinusa:
326 \psline(1.22,2.5)(1.22,2.5042288)
327 \psline(1.22,2.5042288)(1.534,2.812249)
328 \psline(1.534,2.812249)(1.848,3.0902183)
329 \psline(1.848,3.0902183)(2.162,3.3106391)
330 \psline(2.162,3.3106391)(2.476,3.4518504)
331 \psline(2.476,3.4518504)(2.79,3.4999998)
332 \psline(2.79,3.4999998)(3.104,3.4503665)
333 \psline(3.104,3.4503665)(3.418,3.307798)
334 \psline(3.418,3.307798)(3.732,3.0862381)
335 \psline(3.732,3.0862381)(4.046,2.8073506)
336 \psline(4.046,2.8073506)(4.36,2.4984076)
337 \psline(4.36,2.4984076)(4.674,2.1896203)
338 \psline(4.674,2.1896203)(4.988,1.9111845)
339 \psline(4.988,1.9111845)(5.302,1.6903282)
340 \psline(5.302,1.6903282)(5.616,1.5486487)
341 \psline(5.616,1.5486487)(5.93,1.5000004)
342 \psline(5.93,1.5000004)(6.244,1.5491406)
343 \psline(6.244,1.5491406)(6.558,1.691264)
344 \psline(6.558,1.691264)(6.872,1.9124725)
345 \psline(6.872,1.9124725)(7.186,2.1911345)
346 \psline(7.186,2.1911345)(7.5,2.5)
347 \psline(7.5,2.5)(7.814,2.8088655)
348 \psline(7.814,2.8088655)(8.128,3.0875275)
349 \psline(8.128,3.0875275)(8.442,3.308736)
350 \psline(8.442,3.308736)(8.756,3.4508593)
351 \psline(8.756,3.4508593)(9.07,3.4999995)
352 \psline(9.07,3.4999995)(9.384,3.4513514)
353 \psline(9.384,3.4513514)(9.698,3.3096719)
354 \psline(9.698,3.3096719)(10.012,3.0888155)
355 \psline(10.012,3.0888155)(10.326,2.8103797)
356 \psline(10.326,2.8103797)(10.64,2.5015924)
357 \psline(10.64,2.5015924)(10.954,2.1926494)
358 \psline(10.954,2.1926494)(11.268,1.9137619)
359 \psline(11.268,1.9137619)(11.582,1.692202)
360 \psline(11.582,1.692202)(11.896,1.5496336)
361 \psline(11.896,1.5496336)(12.21,1.5000004)
362 \psline(12.21,1.5000004)(12.524,1.5481497)
363 \psline(12.524,1.5481497)(12.838,1.6893607)
364 \psline(12.838,1.6893607)(13.152,1.9097816)
365 \psline(13.152,1.9097816)(13.466,2.187751)
366 \psline(13.466,2.187751)(13.78,2.4957712)
367
368
369 \end{pspicture}
370 \linespread{1}
371 \caption{Układ współrzędnych z funkcją sinus \protect \\\
372 \emph{Źródło: opracowanie własne}}
373 \label{sinus}
374 \end{figure}

```

```

375
376 %-----
377 \subsubsection{Graficzne rozwiązanie układu nierówności}
378
379 Następny przykład zawiera zakreślony obszar,
380 będący rozwiązaniem układu nierówności:
381 $$\left\{
382 \begin{array}{c}
383 \frac{x^2}{5} < y \ \backslash \\
384 -x^2 + 2x + 2 > y \\
385 \end{array} \right. $$
386
387 % Kod generujący rysunek:
388 \begin{figure}[h]
389 \centering
390 \psset{unit=0.8cm}
391 \begin{pspicture}(10,11)
392
393 % Jednostki na osi y:
394 \put(5,0){\vector(0,1){11}}
395 \put(4.9,0){\line(1,0){0.2}} \put(0.1,-0.07){\scriptsize{-5}}
396 \put(4.9,1){\line(1,0){0.2}} \put(0.1,-0.07){\scriptsize{-4}}
397 \put(4.9,2){\line(1,0){0.2}} \put(0.1,-0.07){\scriptsize{-3}}
398 \put(4.9,3){\line(1,0){0.2}} \put(0.1,-0.07){\scriptsize{-2}}
399 \put(4.9,4){\line(1,0){0.2}} \put(0.1,-0.07){\scriptsize{-1}}
400 \put(4.9,6){\line(1,0){0.2}} \put(0.1,-0.07){\scriptsize{1}}
401 \put(4.9,7){\line(1,0){0.2}} \put(0.1,-0.07){\scriptsize{2}}
402 \put(4.9,8){\line(1,0){0.2}} \put(0.1,-0.07){\scriptsize{3}}
403 \put(4.9,9){\line(1,0){0.2}} \put(0.1,-0.07){\scriptsize{4}}
404 \put(4.9,10){\line(1,0){0.2}} \put(0.1,-0.07){\scriptsize{5}}
405
406 \put(5.1,11){$y$}
407
408 % Jednostki na osi x:
409 \put(0,5){\vector(1,0){11}}
410 \put(0,4.9){\line(0,1){0.2}} \put(-0.1,0.25){\scriptsize{-5}}
411 \put(1,4.9){\line(0,1){0.2}} \put(-0.1,0.25){\scriptsize{-4}}
412 \put(2,4.9){\line(0,1){0.2}} \put(-0.1,0.25){\scriptsize{-3}}
413 \put(3,4.9){\line(0,1){0.2}} \put(-0.1,0.25){\scriptsize{-2}}
414 \put(4,4.9){\line(0,1){0.2}} \put(-0.1,0.25){\scriptsize{-1}}
415 \put(6,4.9){\line(0,1){0.2}} \put(-0.1,0.25){\scriptsize{1}}
416 \put(7,4.9){\line(0,1){0.2}} \put(-0.1,0.25){\scriptsize{2}}
417 \put(8,4.9){\line(0,1){0.2}} \put(-0.1,0.25){\scriptsize{3}}
418 \put(9,4.9){\line(0,1){0.2}} \put(-0.1,0.25){\scriptsize{4}}
419 \put(10,4.9){\line(0,1){0.2}} \put(-0.1,0.25){\scriptsize{5}}
420
421 \put(11,5.1){$x$}
422 \put(5.2,5.15){\scriptsize{0}}
423
424 % Rysowanie pierwszej paraboli - \ / :
425 \psline(1,8.2)(1,8.2)
426 \psline(1,8.2)(1.2,7.888)
427 \psline(1.2,7.888)(1.4,7.592)
428 \psline(1.4,7.592)(1.6,7.312)
429 \psline(1.6,7.312)(1.8,7.048)

```

```

430 \psline(1.8,7.048)(2,6.8)
431 \psline(2,6.8)(2.2,6.568)
432 \psline(2.2,6.568)(2.4,6.352)
433 \psline(2.4,6.352)(2.6,6.152)
434 \psline(2.6,6.152)(2.8,5.968)
435 \psline(2.8,5.968)(3,5.8)
436 \psline(3,5.8)(3.2,5.648)
437 \psline(3.2,5.648)(3.4,5.512)
438 \psline(3.4,5.512)(3.6,5.392)
439 \psline(3.6,5.392)(3.8,5.288)
440 \psline(3.8,5.288)(4,5.2)
441 \psline(4,5.2)(4.2,5.128)
442 \psline(4.2,5.128)(4.4,5.072)
443 \psline(4.4,5.072)(4.6,5.032)
444 \psline(4.6,5.032)(4.8,5.008)
445 \psline(4.8,5.008)(5,5)
446 \psline(5,5)(5.2,5.008)
447 \psline(5.2,5.008)(5.4,5.032)
448 \psline(5.4,5.032)(5.6,5.072)
449 \psline(5.6,5.072)(5.8,5.128)
450 \psline(5.8,5.128)(6,5.2)
451 \psline(6,5.2)(6.2,5.288)
452 \psline(6.2,5.288)(6.4,5.392)
453 \psline(6.4,5.392)(6.6,5.512)
454 \psline(6.6,5.512)(6.8,5.648)
455 \psline(6.8,5.648)(7,5.8)
456 \psline(7,5.8)(7.2,5.968)
457 \psline(7.2,5.968)(7.4,6.152)
458 \psline(7.4,6.152)(7.6,6.352)
459 \psline(7.6,6.352)(7.8,6.568)
460 \psline(7.8,6.568)(8,6.8)
461 \psline(8,6.8)(8.2,7.048)
462 \psline(8.2,7.048)(8.4,7.312)
463 \psline(8.4,7.312)(8.6,7.592)
464 \psline(8.6,7.592)(8.8,7.888)
465 \psline(8.8,7.888)(9,8.2)
466
467
468 % Rysowanie drugiej paraboli - /\ :
469 \psline(3.4,1.24)(3.4,1.24)
470 \psline(3.4,1.24)(3.6,2.24)
471 \psline(3.6,2.24)(3.8,3.16)
472 \psline(3.8,3.16)(4,4)
473 \psline(4,4)(4.2,4.76)
474 \psline(4.2,4.76)(4.4,5.44)
475 \psline(4.4,5.44)(4.6,6.04)
476 \psline(4.6,6.04)(4.8,6.56)
477 \psline(4.8,6.56)(5,7)
478 \psline(5,7)(5.2,7.36)
479 \psline(5.2,7.36)(5.4,7.64)
480 \psline(5.4,7.64)(5.6,7.84)
481 \psline(5.6,7.84)(5.8,7.96)
482 \psline(5.8,7.96)(6,8)
483 \psline(6,8)(6.2,7.96)
484 \psline(6.2,7.96)(6.4,7.84)

```

```

485 \psline(6.4,7.84)(6.6,7.64)
486 \psline(6.6,7.64)(6.8,7.36)
487 \psline(6.8,7.36)(7,7)
488 \psline(7,7)(7.2,6.56)
489 \psline(7.2,6.56)(7.4,6.04)
490 \psline(7.4,6.04)(7.6,5.44)
491 \psline(7.6,5.44)(7.8,4.76)
492 \psline(7.8,4.76)(8,4)
493 \psline(8,4)(8.2,3.16)
494 \psline(8.2,3.16)(8.4,2.24)
495 \psline(8.4,2.24)(8.6,1.24)
496
497
498 % Zakreskowanie rozwiązania:
499 \psset{linewidth=.01mm}
500 \psline(4.3,5.098)(4.3,5.11)
501 \psline(4.4,5.072)(4.4,5.44)
502 \psline(4.5,5.05)(4.5,5.75)
503 \psline(4.6,5.032)(4.6,6.04)
504 \psline(4.7,5.018)(4.7,6.31)
505 \psline(4.8,5.008)(4.8,6.56)
506 \psline(4.9,5.002)(4.9,6.79)
507 \psline(5,5)(5,7)
508 \psline(5.1,5.002)(5.1,7.19)
509 \psline(5.2,5.008)(5.2,7.36)
510 \psline(5.3,5.018)(5.3,7.51)
511 \psline(5.4,5.032)(5.4,7.64)
512 \psline(5.5,5.05)(5.5,7.75)
513 \psline(5.6,5.072)(5.6,7.84)
514 \psline(5.7,5.098)(5.7,7.91)
515 \psline(5.8,5.128)(5.8,7.96)
516 \psline(5.9,5.162)(5.9,7.99)
517 \psline(6,5.2)(6,8)
518 \psline(6.1,5.242)(6.1,7.99)
519 \psline(6.2,5.288)(6.2,7.96)
520 \psline(6.3,5.338)(6.3,7.91)
521 \psline(6.4,5.392)(6.4,7.84)
522 \psline(6.5,5.45)(6.5,7.75)
523 \psline(6.6,5.512)(6.6,7.64)
524 \psline(6.7,5.578)(6.7,7.51)
525 \psline(6.8,5.648)(6.8,7.36)
526 \psline(6.9,5.722)(6.9,7.19)
527 \psline(7,5.8)(7,7)
528 \psline(7.1,5.882)(7.1,6.79)
529 \psline(7.2,5.968)(7.2,6.56)
530 \psline(7.3,6.058)(7.3,6.31)
531
532
533 \end{pspicture}
534 \linespread{1}
535 \caption{Rozwiązanie układu nierówności \protect \\
536 \emph{Źródło: opracowanie własne}}
537 \end{figure}

```

D. Dokument prezentacyjny – gotowy dokument

Uwaga: Dokument ten zaleca się czytać wraz z przeglądaniem jego kodu źródłowego, gdyż to tam widoczne są wszystkie przykłady składni systemu $\text{\LaTeX}$.

D.1. Przedstawienie składni systemu $\text{\LaTeX}$

Działanie systemu opiera się na, „wtrąconych” w dokument $\text{\LaTeX}$ -a, blokach instrukcji w języku $\text{\LaTeX}$. Bloki te wyróżnione są w pliku źródłowym $\text{\LaTeX}$ -a znakami: $\langle \# \text{ i } \# \rangle$. Instrukcje w takim bloku muszą kończyć się znakiem $;$ inaczej zostaną pominięte (bez ostrzeżenia).

Deklaracja zmiennych i działania na liczbach

Identyfikatory zmiennych zaczynają się od znaku $\&$. Deklaracja zmiennych w języku $\text{\LaTeX}$ odbywa się niejawnie – poprzez pierwsze przypisanie im wartości.

W wyrażeniach arytmetycznych dostępne są operatory:

- przypisania: $=$, $+=$, $-=$, $*=$, $/=$,
- działań arytmetycznych: $+$, $-$, $*$, $/$, $\%$, $\wedge$,
- inkrementacji: $++$, $--$.

W tym miejscu następuje zadeklarowanie dwóch zmiennych – $a = 1$ i $b = a + 2 = 3$: [Koniec deklaracji] Deklaracja – przypisanie jest „cichą” instrukcją – nie daje widocznych efektów w pliku wynikowym.

Działania na liczbach, nie użyte w przypisaniu, dają wynik w pliku wyjściowym. Przykłady działań na zadeklarowanych powyżej zmiennych:

- $a + b = 4$
- $2b - (a + 3a) = 2$
- $(-ba)/6 = -0.5$

- $b^{((b+2a) \bmod \frac{2b}{2a})} = 9$

Możliwe jest też deklarowanie zmiennych tablicowych. W $\text{K}\text{T}_{\text{E}}\text{X}$ -u są to zmienne zawierające w swojej nazwie liczbę w nawiasach: `[, ]`. Przykłady składni dla zmiennych tablicowych/indeksowanych:

- `tab[7]` (`= 13`): 13
- `tab[7][9]` (`=tab[7]+7`): 20
- `tab[7,9]` (`= tab[7][9]+tab[7]`): 33
- `zmienna[7]numerowana` (`= 23`): 23

Instrukcja warunkowa i operacje logiczne

Składnia instrukcji warunkowej w $\text{K}\text{T}_{\text{E}}\text{X}$ -u ma postać:

```
<#if{warunek}
```

tekst i/lub instrukcje w blokach

```
if#> <#else ... else#>
```

W warunku, można stosować:

- operatory porównania: `==`, `=>`, `=<`, `<`, `>`, `<>`,
- operatory logiczne: negacja – `!`, alternatywa – `||`, koniunkcja – `&&`.

Instrukcje *if* można dowolnie zagnieżdżać.

Przykład działania instrukcji warunkowych:

IF1: Prawda: 7 jest większe od 1 i mniejsze od 13

IF2: Else: 7 jest równe „7”

Pętla *for*

W systemie $\text{K}\text{T}_{\text{E}}\text{X}$ zaimplementowano pętlę *for*. Definiuje się ją specjalnym blokiem:

```
<#for{&zmienna_sterująca:od:krok:do}
```

tekst i/lub instrukcje w blokach

```
for#>
```

Jak widać powyżej, przyjęto deklarację z podaniem nazwy zmiennej sterującej i jej zakresu wraz z krokiem inkrementacji (lub dekrementacji). Parametry *od*, *krok* i *do* są, tak jak wszystkie zmienne w $\text{K}\text{T}_{\text{E}}\text{X}$, typu float. W ich miejscu można stosować także wyrażenia i wywołania funkcji. Pętle można zagnieżdżać oraz wpływać wewnątrz nich na zmienne sterujące.

Przykład efektu działania trzech zagnieżdżonych pętli:

(i, j, k) = (2, 4, 0)

(i, j, k) = (4, 4, 1)

(i, j, k) = (6, 4, 2)

(i, j, k) = (8, 2, 0)

(i, j, k) = (10, 2, 1)

(i, j, k) = (12, 0, 0)

Pętla *for* działająca jak pętla *do/while* (niezależna od pętli zmienna jest zwiększana dopóki jest mniejsza od 10): 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, Przerwanie!

Deklaracja i użycie funkcji

Deklaracji funkcji dokonuje się za pomocą specjalnego bloku:

```
<#function:nazwa{&arg1 , &arg2, ...}
```

kod funkcji

```
function#>
```

Funkcji nie można deklarować wewnątrz pętli ani instrukcji warunkowych. Wewnątrz bloku funkcji może znajdować się zwykły tekst (zostanie przepisany do pliku wynikowego w momencie wywołania funkcji) oraz – przede wszystkim – bloki instrukcji. Wywołanie odbywa się poprzez podanie nazwy poprzedzonej znakiem „-” i argumentów w nawiasach: [i], oddzielonych przecinkami – np.:

`_nazwa[arg1, arg2]`.

Standardowe użycie funkcji wymaga, aby w kodzie nie wypisywano tekstu (spacje też są liczone jako tekst, ale tabulatory nie – są automatycznie pomijane), a wypisana została tylko jedna liczba – wynik funkcji.

W tym miejscu zostanie zadeklarowana funkcja *dodaj*, która dodaje do siebie dwa pobierane argumenty i podaje wynik: [koniec]

Tutaj wywołanie funkcji *dodaj* dla obu argumentów równych 2: 4.

Taką funkcję można używać w wyrażeniach arytmetycznych np. `2^(_dodaj[1,2])`: 8.

Inne użycia (nie mogą występować w wyrażeniach):

- jako procedura – wykonuje obliczenia, zmienia wartości zmiennych (poprzez przy-

pisanie), ale sama nic nie wypisuje.

Przykład:

Zadeklarowanie funkcji bezargumentowej ustalającej zmienne $a = 7$ i $b = 13$:

[koniec]

Wywołanie: [koniec]. Wypisanie zmiennych a i b : 7 13.

- jako makro – mogące wypisywać tekst, wartości zmiennych itp.

Przykład:

Zadeklarowanie funkcji pobierającej liczbę i wypisującej ją wraz z tekstem: [koniec]

Wywołanie: Podana liczba to: 100

D.2. Praktyczne przykłady zastosowań K_TE_X-a

Tabela z sumą i średnią wartością kolumny

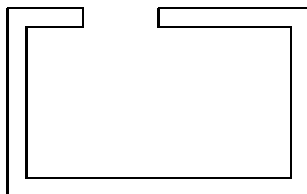
K_TE_X pozwala również na wykonywanie obliczeń i wykorzystanie ich wyników jako zwykły tekst. Poniżej pokazano przykład tabeli, w której policzona została suma i średnia wartość kolumny. Zmieniając jedną z wartości w pliku źródłowym wynik zostanie zaktualizowany.

Tabela D.1. Przykład tabeli z zastosowaniem K_TE_X-a
Dane na podstawie: Wikipedia – List of countries by carbon dioxide emissions
http://en.wikipedia.org/wiki/List_of_countries_by_carbon_dioxide_emissions

Pozycja na świecie	Państwo	Roczna emisja CO ₂ w milionach ton
1	USA	6049
2	Chiny i Tajwan	5010
3	Rosja	1524
4	Indie	1342
5	Japonia	1257
6	Niemcy	860
7	Kanada	639
8	W. Brytania	587
9	Pln. Korea	465
10	Włochy	449
...		
21	Polska	307
Suma:		18489
Średnia:		1680.8182

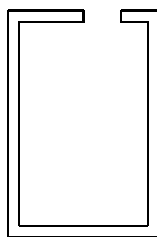
Plan pomieszczenia, ze zmianą proporcji

W rozdziale 3.2 przedstawiono prosty rysunek pokoju (3.1) i wykazano trudność w jego aktualizacji – zmianie położenia drzwi, czy zmniejszenie grubości ścian. Poniżej znajduje się deklaracja funkcji K_TE_X-a, która rysuje ten plan zgodnie z podanymi parametrami: szerokość, długość, pozycja drzwi, szerokość drzwi i grubość ściany. Następnie wywołano ją trzy razy z różnymi argumentami.



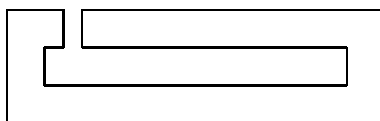
Rysunek D.1. Plan pomieszczenia nr. 1

Źródło: opracowanie własne



Rysunek D.2. Plan pomieszczenia nr. 2

Źródło: opracowanie własne

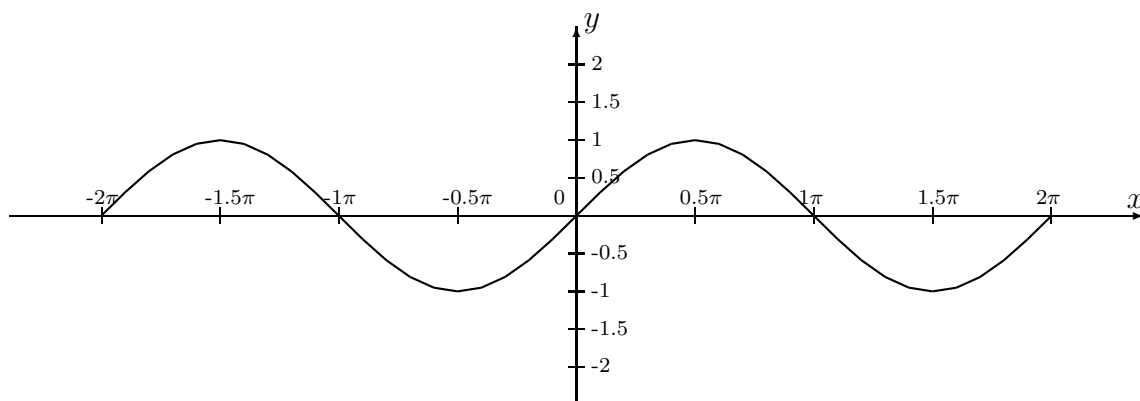


Rysunek D.3. Plan pomieszczenia nr. 3

Źródło: opracowanie własne

Układ współrzędnych z funkcją sinus

W tym przykładzie zastosowano K_TE_X-a do wypisania jednostek na osiach oraz do wygenerowania ciągu przybliżonych wartości sinusa (z szeregu Taylora) – za pomocą zaprogramowanych funkcji, które posłużyły do narysowania wykresu. Wynik działania obrazuje rysunek D.4.



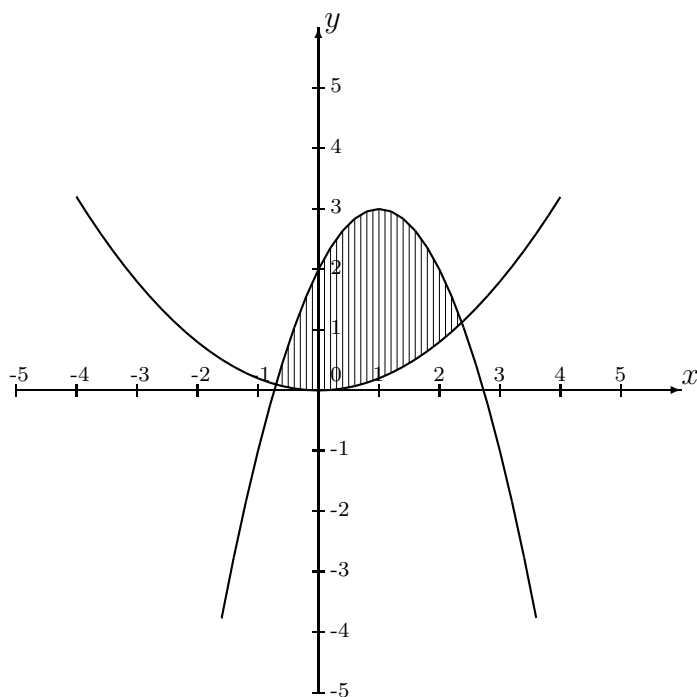
Rysunek D.4. Układ współrzędnych z funkcją sinus

Źródło: opracowanie własne

Graficzne rozwiązanie układu nierówności

Następny przykład zawiera zakreślony obszar, będący rozwiązaniem układu nierówności:

$$\begin{cases} \frac{x^2}{5} < y \\ -x^2 + 2x + 2 > y \end{cases}$$



Rysunek D.5. Rozwiązanie układu nierówności

Źródło: opracowanie własne

E. Zawartość CD

1. Dokument PDF z tekstem pracy – plik: `praca.pdf`.
2. Kod źródłowy systemu K_T_EX – katalog: `ktex_source`.
3. Skompilowany program K_T_EX – katalog: `ktex`.
4. Dokument prezentacyjny K_T_EX-a – w katalogu `ktex_dok`:
 - `ktex_dok.ktex` – plik źródłowy K_T_EX,
 - `ktex_dok.tex` – plik przetworzony K_T_EX-em,
 - `ktex_dok.pdf` – gotowy dokument.