Pomiary Automatyka Robotyka, ISSN 1427-9126, R. 24, Nr 2/2020, 11–16, DOI: 10.14313/PAR_236/11

# O znaczeniu odwrotnej notacji polskiej dla rozwoju technik informatycznych

Sławomir Gryś, Waldemar Minkina
Politechnika Częstochowska, Wydział Elektryczny, Aleja Armii Krajowej 17, 42-218 Częstochowa

Streszczenie: Artykuł dotyczy wpływu beznawiasowej notacji polskiej na rozwój współczesnej informatyki i innych, pokrewnych dyscyplin naukowych korzystających z technik informatycznych, m.in. matematyki, elektroniki, automatyki czy robotyki. W niniejszym artykule podano przykłady współczesnych zastosowań ONP w informatyce, m.in. w językach Forth, Postscript i parserach języków wysokiego poziomu. Rozważania zilustrowano przykładami konwersji wyrażeń z klasycznej notacji do notacji przyrostkowej oraz ich przetwarzania przez komputer z użyciem modelu zarządzania pamięcią zorganizowaną w stos, dzięki czemu uzyskuje się skrócenie czasu wykonania algorytmu i mniejszą zajętość pamięci. Celem tej publikacji było także upamiętnienie osiągnięć polskich naukowców w zakresie współczesnych technik komputerowych, a w szczególności Jana Łukasiewicza – twórcy znanej na całym świecie Odwrotnej Notacji Polskiej. Stworzona obecnie w Polsce sieć badawcza obejmująca wiodące krajowe instytuty naukowo-badawcze przyjęła patronat innego badacza: Ignacego Łukasiewicza – znanego głównie z wynalezienia lampy naftowej i rozwoju polskiego przemysłu naftowego. Warto pamiętać, że są to różni naukowcy zasłużeni w różnych obszarach badawczych.

**Słowa kluczowe:** historia informatyki, notacja Łukasiewicza, Odwrotna Notacja Polska, wyrażenia logiczne i algebraiczne, stos w komputerze, języki programowania

## 1. Wprowadzenie

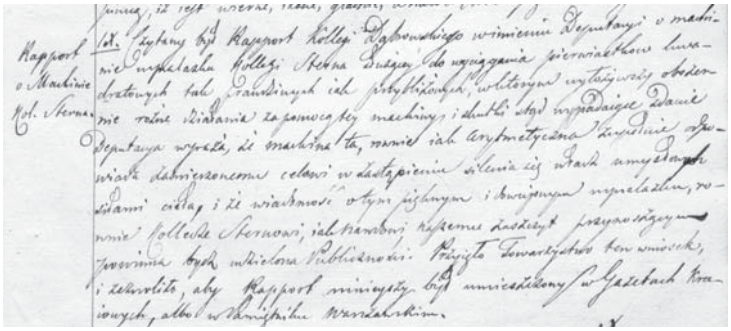
Odwrotna Notacja Polska została opracowana i upowszechniona przez australijskiego naukowca Charlesa Hamblina [1] jako „odwrócenie" beznawiasowej notacji polskiej, opracowanej około 1920 r. przez polskiego matematyka Jana Łukasiewicza. Zarówno odkrycie, jak i jego fundamentalne znaczenie dla rozwoju technologii cyfrowej nie są powszechnie znane nawet wśród osób z wykształceniem akademickim. Działanie wszelkich urządzeń zawierających procesory – kalkulatory, komputery, urządzenia sieciowe i telekomunikacyjne, telefony komórkowe, sprzęt AGD, cyfrowa telewizja naziemna i satelitarna i inne urządzenia – byłoby bardzo utrudnione, gdyby nie stosowano tej notacji. Jej znaczenie doceniono dopiero po wielu dekadach w epoce komputerów wyposażonych w programowalny procesor i wykorzystujących języki wysokiego poziomu. Zaletą tej notacji jest brak nawiasów oraz konieczności wstępnej analizy całego wyrażenia arytmetycznego, aby ustalić kolejność działań. Taki sposób zapisu ułatwia przygotowanie kodu wynikowego dla procesora. Notacja może być stosowana do zapisu dowolnych wyrażeń złożonych z symboli reprezentujących zmienne wejściowe oraz operatorów, a więc w logice zdań, teorii zbiorów, algebrze, logice dwuwartościowej będącej podstawą działania układów cyfrowych i komputerów, a także w translatorach języków wysokiego poziomu.

W przypisie do pracy [2] Jan Łukasiewicz po raz pierwszy podał reguły stosowanego przez siebie zapisu, zwanego także zapisem przedrostkowym, a w literaturze anglojęzycznej określanego jako *Polish Notation* (*PN*), *Normal Polish Notation* (*NPN*), **Łukasiewicz Notation** (**ŁN**), *Warsaw Notation* (*WN*), *Polish Prefix Notation* (*PPN*), *Simply Prefix Notation* (*SPN*). Hamblin sugerował, aby notację tę nazwać „Azciweisakul notation" (od nazwiska pisanego od tyłu). Ta propozycja nie przyjęła się prawdopodobnie ze względu na trudności językowe.

Przybliżmy sylwetkę prof. Jana Łukasiewicza. Urodził się 21 grudnia 1878 r. we Lwowie, a zmarł 13 lutego 1956 r. w Dublinie. Zajmował się zawodowo logiką, matematyką i filozofią, pracując na Uniwersytecie Jana Kazimierza we Lwowie, Uniwersytecie Warszawskim, a za granicą – na Uniwersytecie Dublińskim od 1946 r. Pełnił funkcję rektora Uniwersytetu Warszawskiego (1922–23 i 1931–32) oraz ministra oświaty w rządzie Ignacego Paderewskiego. Jest wymieniony wśród grona twórców tzw. *polskiej lwowsko-warszawskiej szkoły matematycznej*. Do jego osiągnięć zalicza się położenie podwalin teoretycznych pod logikę trójwartościową, w której oprócz dwóch klasycznych stanów *prawda* i *fałsz* trzeci stan oznaczono jako *nie wiem*. Choć współczesne komputery nie korzystają dziś z zalet takiej konwencji, warto wspomnieć, że w drugiej połowie XX wieku w Związku Radzieckim produkowano maszyny cyfrowe SETUN (1958), oparte na podobnym pomyśle, oznaczając symbolicznie stany 1, 0, –1, co nadało im znaczenie arytmetyczne [3].

**Autor korespondujący:**
Sławomir Gryś, grys@el.pcz.czest.pl

**Artykuł recenzowany**
nadesłany 08.05.2020 r., przyjęty do druku 26.06.2020 r.



**Rys. 1. Fragment sprawozdania z 13 kwietnia 1817 r. Towarzystwa Przyjaciół Nauk [4]**
Fig. 1. Excerpt from the report of April 13, 1817 of the Society of Friends of Sciences [4]

Poruszając wątek wpływu naszych rodaków na rozwój informatyki, nie wypada przemilczeć wynalazców żyjących w XIX wieku, inspirujących kolejne pokolenia inżynierów. Przeszli oni do historii techniki jako twórcy mechanicznych maszyn liczących. Byli to Abraham Stern (176?–1842), Izrael Abraham Staffel (1814–1884) i Jewna Jakobson żyjący w drugiej połowie XIX wieku. Na rys. 1 przedstawiono fragment sprawozdania z posiedzenia Towarzystwa Przyjaciół Nauk z dnia 13 kwietnia 1817 r., na którym dyskutowano o znaczeniu maszyny pierwiastkującej Sterna [4]. Czytamy w nim:

„..."*IX. Raport o Machinie Kollegi Sterna*

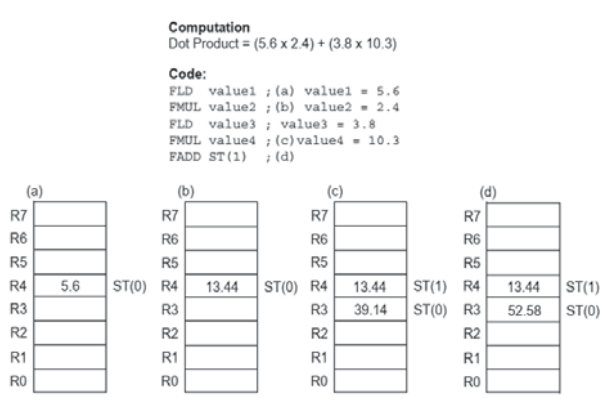
*Czytany był Rapport Kollegi Dąbrowskiego w imieniu Deputacyi o machinie wynalazku Kollegi Sterna służącej do wyciągania pierwiastków kwadratowych tak prawdziwych jak przybliżonych, w którym wyłożywszy ‚obozennie' różne działania za pomocą tej machiny i skutki z tych wypadające zdanie Deputacya wyraża, że machina ta, równie jak arytmetyczna zupełnie odpowiada zamierzonemu celowi w zastąpieniu silenia się władz umysłowych siłami ciała, i że wiadomość o tym pięknym i '...' wynalazku, równie Kolledze Sternowi, jak narodowi naszemu zaszczyt przynoszącym powinna być udzielona Publiczności. Przyjęło Towarzystwo ten wniosek, i zezwoliło, aby Raport niniejszy był umieszczony w Gazetach krajowych, albo w Pamiętniku Warszawskim."*

Warto przypomnieć, że komputery uzyskały zdolność obliczania pierwiastka kwadratowego dopiero nieco ponad pół wieku temu. W latach 60. XX wieku były dostępne m.in. maszyny IBM 360/370, DEC VAX czy polska ODRA 1003/1013. Jednak pierwiastkowanie stało się podstawową funkcjonalnością komputera wraz z opracowaniem w 1982 r. przez firmę Intel układu koprocesora arytmetycznego, zwanego też jednostką zmiennopozycyjną, a trzy lata później wprowadzono światowy standard IEEE Standard for Floating-Point Arithmetic (IEEE 754 – 1985). W 2008 r. wprowadzono zmiany do tej normy, dostosowując jej zapisy do współczesnych potrzeb. Zastosowano m.in. format 128-bitowy dla zadań o wymaganej przez środowisko naukowców bardzo wysokiej rozdzielczości (precyzji) zapisu liczb. Mając na uwadze rynek tanich procesorów zalegalizowano format 16-bitowy, w pełni wystarczający dla przenośnych urządzeń rejestrująco-fiskalnych (parkometry, biletomaty, urządzenia odczytujące stan liczników zużycia wody, gazu, energii elektrycznej itp.).

## 2. Notacja polska

Zgodnie z notacją zaproponowaną przez Jana Łukasiewicza, w zapisie wyrażeń arytmetycznych najpierw podawany jest operator (działanie), a potem operandy (argumenty). Wyrażenie nie wymaga wówczas użycia nawiasów, ponieważ przypisanie argumentów do operatorów wynika wprost z ich kolejności w zapisie, o ile z góry znana jest liczba argumentów poszczególnych operatorów. Przy takich założeniach nie jest konieczna wstępna analiza całego wyrażenia, aby ustalić kolejność działań. Umożliwia to szybkie wykonanie kodu wynikowego przez procesor z jednoczesną jednokrotną analizą w fazie kompilacji programu każdego z symboli tworzących wyrażenie. Przykładowy zapis działania: $(4+7) \cdot 2$ w notacji klasycznej – wrostkowej (operator umieszczony między liczbami), w zapisie zgodnym z notacją polską wygląda następująco: $\cdot + 4\ 7\ 2$. Taki zapis jest bliski naturalnemu sposobowi wyrażania opisanych działań, w którym to sposobie zazwyczaj najpierw podaje się czynność, a następnie dopełnia wyrażenie wskazaniem rzeczy, do których czynność się odnosi. Mówimy wówczas: „pomnóż sumę 4 i 7 przez 2". Taka interpretacja językowa rozwiewa też wątpliwości, co do kolejności argumentów w przypadku operatorów nieprzemiennych, typu odejmowanie czy dzielenie. Wyżej wymienione cechy naturalności języka przeniesiono również na poziom konstruowania programów dla komputerów w językach niskiego poziomu. Każdy procesor wykonuje program będący sekwencją elementarnych rozkazów zdefiniowanych przez właściwą dla niego listę rozkazów ISA (ang. *Instruction Set Architecture*). Przykładowo, zgodnie z konwencją zaproponowaną przez firmę Intel składnię rozkazu *MOV A*, *B* czytamy jako „prześlij kopię danych z B do A". Inny rozkaz *ADD A*, *B* oznacza: „zapisz w A sumę zawartości z A i B", czyli wskazuje na zapis algebraiczny oznaczany jako A = A + B. Na rysunku 2 przedstawiono fragment rzeczywistego kodu wykonywanego przez koprocesor arytmetyczny x87 zaczerpnięty z dokumentacji technicznej [5]. Koprocesor oblicza wyrażenie $(5{,}6 \cdot 2{,}4) + (3{,}8 \cdot 10{,}3)$. Odpowiadający mu zapis w notacji polskiej ma postać: $+ \cdot 5{,}6\ 2{,}4 \cdot 3{,}8\ 10{,}3$, który jednak nie jest już taki intuicyjny. Zadanie staje się łatwiejsze, jeśli korzysta się ze stosu zorganizowanego z fizycznych rejestrów o symbolach R0–R7, jak w przykładzie poniżej. Wówczas wystarczy w rozkazie podać drugi argument działania, a zarówno jego wynik, jak i pierwszy argument jest wskazywany przez wierzchołek stosu oznaczony tu jako ST(0), co oznacza adresowanie z zerowym przesunięciem względem rejestru pełniącego rolę wierzchołka stosu. Użycie organizacji pamięci w stos przynosi korzyści, gdyż program przetwarza wówczas liczby w naturalnej kolejności zaczynając od liczby 5,6 i zajmuje stosunkowo niewiele zasobów pamięciowych – tylko dwa rejestry! Jako ostatnie jest wykonywane dodawanie, rozkazem FADD, na argumencie przechowywanym na wierzchołku stosu ST(0) i leżącym bezpośrednio pod nim, czyli w lokacji ST(1). Należy dodać, że stos w tym przypadku jest zorganizowany w stronę malejących adresów, czyli „rośnie w dół".

Cecha przetwarzania dwóch argumentów leżących najbliżej wierzchołka stosu, wraz z jego zastosowaniem do przechowywania pośrednich wyników obliczeń, przyczyniła się do upowszechnienia **odwrotnej notacji polskiej**, o której piszemy w kolejnym rozdziale. Więcej rozważań na temat klasycznej arytmetyki komputera, w tym obliczeń zmiennopozycyjnych wraz z wyjaśnieniem podstawowych pojęć związanych z architekturą komputera można znaleźć między innymi w pracach [6, 7].




**Rys. 2. Przykład przetwarzania wyrażenie przez jednostkę FPU [5]**
Fig. 2. Example x87 FPU Dot Product Computation [5]

## 3. Odwrotna notacja polska

Gdy umieści się operatory działań arytmetycznych za operandami, to otrzymuje się tzw. odwrotną notację polską, często nazywaną notacją przyrostkową. Dla uproszczenia dalszego wywodu przywołamy przykład liczbowy z poprzedniego podrozdziału uproszczony, dla lepszej jego czytelności, do liczb z odciętą częścią ułamkową tj. $(5 \cdot 2) + (3 \cdot 10)$. W konwencji ONP zapis prezentuje się następująco: 5 2 · 3 10 · +. Z formalnego punktu widzenia odwrócenie konwencji nie wnosi nic nowego do rozważań. Jeśli jednak założymy, że do przetwarzania wyrażeń w ONP wykorzystuje się komputery, to korzyść staje się widoczna. W notacji polskiej musielibyśmy przechowywać w pamięci komputera operator do momentu wczytania obu argumentów. Obciąża to niepotrzebnie procesor, co ma wpływ na wydajność przetwarzania algorytmów obliczeniowych. W odwrotnej notacji polskiej najpierw wczytujemy argumenty i wykonujemy na nich wskazane działanie. W praktyce w tym celu korzysta się ze stosu wg zasady, że przetwarzane jest jednokrotnie wyrażenie od lewej do prawej i jeśli analizowany element jest:

— argumentem (np. liczbą), to zapisywany jest na stosie,
— operatorem, to pobierane są ze stosu dwa argumenty, wykonywane jest działanie, a wynik zapisywany na stosie.

Wykonanie naszego wyrażenia w ONP z użyciem stosu, zorganizowanego „w dół", można zilustrować jak na rysunku 3, wyróżniając analizowany w danym etapie element (ang. *token*). Przetwarzanie w sposób opisany powyżej wymaga wcześniejszej konwersji wyrażenia do postaci przyrostkowej. W tym celu również można ponownie skorzystać ze stosu.

Wyrażenie w notacji wrostkowej (mogą być stosowane wielopoziomowe nawiasy) przetwarza się jednokrotnie od lewej do prawej wg następującej reguły:

Jeśli analizowany element jest:

— nawiasem otwierającym, to zapisz go na stosie,
— nawiasem zamykającym, to odczytaj ze stosu, w odwrotnej kolejności, wszystkie operatory włącznie z nawiasem otwierającym i dopisz je do wyrażenia w ONP,
— argumentem, to dopisz go do wyrażenia w ONP,
— operatorem, jeśli ma on wyższy priorytet niż ten leżący na wierzchu stosu, to zapisz go na stosie,
— operatorem, jeśli ma niższy lub równy priorytet niż ten leżący na wierzchu stosu, to zapisz go na stosie po odczytaniu operatorów o wyższym lub równym priorytecie,
— na koniec odczytaj ze stosu pozostałe operatory i dopisz do wyrażenia w ONP.

Opisany algorytm jest znany w literaturze jako metoda „Shunting-yard" zaproponowana przez Edgara Dijkstrę. Więcej rozważań na temat różnych algorytmów i ich modyfikacji można znaleźć w pracy [8]. Poszczególne etapy konwersji ww. wyrażenia ilustruje rys. 4.

W przykładzie tym ponownie ujawnia się zaleta stosowania stosu. Do przeprowadzenia konwersji wyrażenia było konieczne tymczasowe użycie jedynie trzech komórek pamięci.

| Etap | Stos | Wyrażenie w ONP |
|---|---|---|
| 1 | ( | |
| 2 | ( | **5** |
| 3 | ( · | 5 |
| 4 | ( · | 5 **2** |
| 5 | | 5 2 · |
| 6 | + | 5 2 · |
| 7 | +( | 5 2 · |
| 8 | +( | 5 2 · **3** |
| 9 | +( · | 5 2 · 3 |
| 10 | +( · | 5 2 · 3 **10** |
| 11 | + | 5 2 · 3 10 · |
| 12 | | 5 2 · 3 10 · + |

**Rys. 4. Konwersja wyrażenia wrostkowego do notacji ONP z użyciem stosu**
Fig. 4. Conversion of infix to postfix notation using the stack

## 4. Zastosowania odwrotnej notacji polskiej

Na przestrzeni lat zastosowania ONP ewoluowało od kalkulatorów elektronicznych ku informatyce. Obecnie użytkownik komputera właściwie ma niewielkie szanse na bezpośredni kontakt z ONP, chyba że jest użytkownikiem mało popularnych w Polsce kalkulatorów firmy Hewlett-Packard lub specjalistycznych aplikacji.

### 4.1. Kalkulatory

Istotna popularyzacja notacji nastąpiła wraz z pojawieniem się kalkulatorów firmy Hewlett-Packard oraz CASIO w latach 70. oraz 80. XX wieku, często nie do końca legalne sprowadzanych z Wielkiej Brytanii – takie były czasy! Posiadanie dolarów w najlepszym przypadku groziło ich konfiskatą. Wczesne modele, m.in. HP-12c, HP-35, używały ONP bez stosowania alternatywnych metod. Późniejsze wersje, jak HP-35s, również udostępniały klasyczną notację wrostkową i mogły w wygodny sposób umożliwić użytkownikom przełączanie się między nią, a ONP. Rozszerzoną funkcjonalnością, w tym obsługą ONP, charakteryzuje się obecna na rynku linia modeli HP Prime. Innym, powszechnie dostępnym rozwiązaniem są aplikacje wirtualnego kalkulatora z funkcją ONP na urządzenia mobilne. Przykładową, aktualnie stosowaną w smartfonach aplikację wirtualnego kalkulatora „HiPER Scientific Calculator" przedstawiono na rys. 5.

| Etap 1 | | Etap 2 | | Etap 3 | | Etap 4 | | Etap 5 | | Etap 6 | | Etap 7 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **5** | ST(0) | 5 | ST(1) | **5 · 2** | ST(0) | 5 · 2 | ST(1) | 5 · 2 | ST(2) | 5 · 2 | ST(1) | **5 · 2+3 · 10** | ST(0) |
| | | **2** | ST(0) | | | **3** | ST(0) | 3 | ST(1) | **3 · 10** | ST(0) | | |
| | | | | | | | | **10** | ST(0) | | | | |
| | | | | | | | | | | | | | |

**Rys. 3. Kolejne etapy przetwarzania wyrażenia w ONP z użyciem stosu**
Fig. 3.The next stages of processing RPN expression using the stack


a) b)

**Rys. 5. Widok aplikacji kalkulatora z możliwością wyboru trybu ONP (a) oraz przykładowe liczby odkładane na stosie (b)**
Fig. 5. Virtual calculator for mobile working in RPN mode as one of modes (a) and examples of numbers pushed on stack (b)

## 4.2. Komputery

Naturalnymi następcami kalkulatorów stały się komputery. Niektóre modele potrafiły w sposób natywny przetwarzać wyrażenia ONP, co jest bliskie specyfice komputera i używanych w nim rozkazów. Zaliczały się do nich modele KDF9 firmy English Electric Company i amerykańskie Burroughs B5000, popularne w latach 60. XX wieku. Dzisiejsze komputery, bez których nie wyobrażamy sobie współczesnego świata, nie działałyby tak efektywnie, gdyby nie modyfikacja pomysłu Jana Łukasiewicza. Wszelkie wyrażenia w zrozumiałej dla człowieka postaci wrostkowej (a może to tylko kwestia szkolnych nawyków?) w tle są konwertowane do beznawiasowej postaci przyrostkowej. Nietrudno wskazać liczne programy wspierające matematyków, fizyków, czy inżynierów, umożliwiające operowanie na zapisie (niekiedy symbolicznym) wyrażeń algebraicznych, logicznych i leksykalnych. Konwersja do ONP jest przeprowadzana podczas wczesnej fazy kompilacji programów napisanych w popularnych językach wysokiego poziomu w trakcie analizy składni leksykalnej kodu źródłowego, tzw. parsery. Istnieją też takie języki, dla których ONP jest natywnym sposobem opisu wyrażeń.

## 4.3. Postscript

Jest to uniwersalny język opisu strony przygotowanej do druku, będący obecnie standardem w zastosowaniach poligraficznych – cechuje go rozszerzenie pliku *.PS. Postscript jest równocześnie proceduralnym językiem programowania, opartym na strukturze stosu. Pozwala nie tylko opisać precyzyjnie wygląd strony, *łącząc obiekty tekstowe z graficznymi,* ale także wykonywać złożone operacje na dostarczonych danych. Jego polecenia są rozpoznawane przez urządzenia drukujące. W składni języka używana jest odwrotna notacja polska, co sprawia, że kolejność operacji jest jednoznaczna, ale wpływa to niekorzystnie na czytelność programu, ponieważ analizując kod należy pamiętać o bieżącej zawartości stosu. Większość operatorów pobiera argumenty ze stosu i umieszcza wyniki na stosie. Literały, na przykład liczby, powodują umieszczenie ich kopii na stosie.

## 4.4. Język Forth

Jest językiem bliskim sprzętowi, stąd potocznie nazywany jest asemblerowym językiem programowania wysokiego poziomu. Wynika to z wbudowanych cech paradygmatu imperatywnego i silnego zorientowania na wykonywanie operacji na stosie. Do jego charakterystycznych cech należą: natychmiastowa interpretacja wprowadzonych słów oraz to, co jest kluczowe w kontekście niniejszego artykułu – wykonywanie kodu w konwencji odwrotnej notacji polskiej *właśnie z użyciem stosu.* Oparcie języka Forth na ONP wyraźnie wyróżnia go spośród wielu innych koncepcji tworzenia języków programowania. Początkowa trudność w zrozumieniu kodu przez niedoświadczonych programistów prawdopodobnie stała się przyczyną ograniczonej popularności tego języka, choć istnieje wiele jego wersji na różne platformy sprzętowe i systemy operacyjne. Warto wymienić kilka z nich, m.in. Swiftforth, Gforth, Mecrisp.

## 4.5. Projektowanie układów logicznych

Stosując formalizm naukowy można stwierdzić, że komputery w swojej naturze są niczym innym jak sekwencyjną maszyną synchroniczną, czyli układem przechodzącym ze stanu poprzedniego do stanu następnego w takt zegara taktującego. Na stan następny ma wpływ stan poprzedni (zawartość pamięci danych komputera) oraz dane wejściowe, np. uzyskane w trakcie interakcyjnej konwersacji z użytkownikiem lub zawartość pliku. Strukturę komputera można więc opisać jako kombinację funktorów logicznych *i układów pamięciowych.* Na przestrzeni lat do projektowania układów logicznych, powszechnie nazywanych też cyfrowymi, wynaleziono wiele metod optymalizacji wyrażeń boolowskich *służących do opisu architektury* komputera. Wykorzystując go jako narzędzie do projektowania układów logicznych istotne są metody pozwalające na zautomatyzowanie i algorytmizację tego procesu. Jedną z bardziej efektywnych metod jest zaproponowana przez Quinea i McCluskeya [9, 10]. We wczesnej fazie działania algorytmu korzysta ona z właściwości ONP pozbywając się nawiasów i konwertując wejściowe wyrażenie logiczne do postaci przyrostkowej. Upraszcza to kolejne kroki poszukiwania minimalnej postaci opisu formalnego projektowanego układu, na podstawie którego tworzona jest sieć połączeń funktorów logicznych. Przykładowo, proste wyrażenie logiczne $(A \vee B) \wedge C$ jest sprowadzane do równoważnej postaci $AB \vee C \wedge$, gdzie $\vee$ i $\wedge$ to operatory alternatywy i koniunkcji.

# 5. Podsumowanie

Podane przykłady zastosowań odwrotnej notacji polskiej nie wyczerpują tematu, a jedynie są zachętą do dalszego studiowania zagadnienia. Ponieważ niniejszy artykuł w znacznym stopniu dotyka aspektów historii techniki, podkreślając osiągnięcia polskiej myśli wynalazczej, stąd zakończymy go słowami Abrahama Sterna – jednego ze wspomnianych wcześniej Ojców maszyn liczących. Uogólniliśmy jedynie oryginalne znaczenie słowa „mechanika" do słowa „technika", w znaczeniu „informatyki", „automatyki i robotyki", które przecież nie istniały w jego epoce. Poniżej zachowano pisownię oryginalną [11]:

*„... Kończę rozprawę tą uwagą, że gdy mechanika (technika – dopisek S. G. i W. M.) iest klucznicą naszych potrzeb tak dalece, iż nie tylko siłę naszę fizyczną ale nawet i władze umysłowe zastępować może, nayusilnieyszem przeto staraniem naszem bydź powinno przemysł w iey tak obszernem i użytecznem polu rozkrzewiać...*

*...staraymy się ... czynić postęp w przedmiotach, mechanicznych (technicznych – dopisek S. G. i W. M.) możności podlegających, takie bowiem postępowanie drogę do pomyślności i chwały kraiu toruie ...*"

*Abraham Stern – 1817*

Także prace autorów niniejszej publikacji dotyczące tworzenia algorytmów do cyfrowej korekcji dynamicznego przetwarzania przetworników pomiarowych w swojej istocie, nieświadomie, wykorzystywały Odwrotną Notację Polską [6, 12].

## Podziękowanie

Projekt finansowany w ramach programu Ministra Nauki i Szkolnictwa Wyższego pod nazwą „Regionalna Inicjatywa Doskonałości" w latach 2019–2022 nr projektu 020/RID/2018/19, kwota finansowania 12 000 000 PLN.

## Bibliografia


1. Hamblin Ch.L., *Translation to and from Polish notation*. "Computer Journal", Vol. 5, No. 3, 1962, 210–213, DOI: /10.1093/comjnl/5.3.210.
2. Łukasiewicz J., *O znaczeniu i potrzebach logiki matematycznej*, Nauka Polska, 1929, t. 10, 604–620.
3. Targowski A., *Historia – Teraźniejszość – Przyszłość Informatyki*, Wydawnictwo Politechniki Łódzkiej, Łódź 2013, ISBN 978-83-7283-535-2.
4. Archiwum cyfrowe Towarzystwa Przyjaciół Nauk z dnia 13 kwietnia 1817 r., https://szukajwarchiwach.pl/1/199/0/--/057?q=abraham+stern &wynik =6&rpp=15&page =1#tabJednostka.
5. Intel 64 and IA-32 Architectures Software Developer's Manual: Vol. 1: Basic Architecture, www.intel.pl/content/www/pl/pl/architecture-and-technology/64-ia-32-architectures-software-developer-vol-1-manual.html.
6. Gryś S., *Arytmetyka komputerów w praktyce*, Wydawnictwo Naukowe PWN, Warszawa 2007, 2013, ISBN 978-83-01-15131-7.
7. Jakubiec J., Krupanek B., Bogacz R., *Narzędzia programowania mikrokontrolerów*. Wydawnictwo Politechniki Śląskiej, Gliwice, 2017, ISBN 978-83-7880-450-5.
8. Ball J.A., A*lgorithms for RPN calculators*, Cambridge, Massachusetts, USA: Wiley-Interscience, John Wiley & Sons, Inc., 1978, ISBN 0-471-03070-8.
9. Quine W.V., *The Problem of Simplifying Truth Functions*, "The American Mathematical Monthly", Vol. 59, No. 8, 1952, 521–531, DOI: 10.2307/2308219.
10. McCluskey E.J., *Minimization of Boolean Functions*, "Bell System Technical Journal", Vol. 35, No. 6, 1956, 1417–1444, DOI: 10.1002/j.1538-7305.1956.tb03835.x.
11. *Rozprawa o Machinie Arytmetyczney połączoney z machiną do wyciągania pierwiastków z ułomkami; przez Abrahama Stern, na posiedzeniu publicznym Towarzystwa Królewskiego Warszawskiego Przyiacioł Nauk d: 30 Kwietnia 1817. czytana*. Biblioteka Cyfrowa Politechniki Warszawskiej, http://bcpw.bg.pw.edu.pl/Content/2099 /07cnomf_omachinie.pdf – dostęp 25.04.2020 r.
12. Minkina W., Gryś S., *Korekcja charakterystyk dynamicznych czujników termometrycznych – metody, układy, algorytmy*, Wydawnictwo Politechniki Częstochowskiej, Częstochowa 2004, ISBN 83-7193-243-X.


# On the Importance of Reverse Polish Notation for the Development of Computer Science


Abstract: This paper is focused on the impact of polish notation on the development of modern computer science and other related scientific disciplines using IT techniques, including mathematics, electronics, automation and robotics. Reverse Polish Notation was developed and disseminated by Australian scientist Charles Hamblin as a "reversal" of the prefix notation, developed around 1920 by the Polish mathematician Jan Łukasiewicz. According to knowledge of the authors of this article, the discovery and its fundamental importance for the development of digital technology is not common in the consciousness, even of people with academic education. Operation of all devices containing processors, e.g. calculators, computers, network and telecommunications devices, cell phones, household appliances, digital terrestrial and satellite television and other devices would be very difficult without applying this notation. Its significance was appreciated only after many decades, in the era of electronic computers, equipped with a programmable processor and high-level languages. The advantage of this notation is the lack of parentheses and the need for a preliminary analysis of the entire expression to determine the order of operations. It makes easier to prepare the result code for the computer. Notation can be used to write any of the expressions composed of symbols representing input variables and operators, i.e. in sentence logic, set theory, algebra, two-state logic used by digital systems and being the basis of computers, high-level language compilers and interpreters. In this article, examples of contemporary RPN applications are given, such as: Forth language, Postscript, high-level language parsers. The considerations are illustrated by examples of the conversion of an expression from classical notation to postfix notation and its processing by a computer using a stack memory management model to reduce time of algorithm execution and memory occupation. The purpose of this publication was also to commemorate the achievements of Polish technical thought in the field of contemporary computer techniques and closely associated with the name of Jan Łukasiewicz – the creator of the world-famous Reverse Polish Notation. This is also due to the fact that the currently created research network including leading national scientific and research institutes in Poland, which is a showcase of Polish technical thought, has assumed the patronage of Ignacy Łukasiewicz – known mainly for the invention of the kerosene lamp and the development of the Polish oil industry. They are two different people distinguished in other areas, which are worth remembering.


**Keywords:** history of computer science, Lukasiewicz notation, Reverse Polish Notation, logical and algebraic expressions, stack, programming languages

## dr hab. inż. Sławomir Gryś, prof. PCz

slawomir.grys@pcz.pl

**ORCID: 0000-0001-7910-3659**


Uzyskał tytuł zawodowy magistra i stopień doktora nauk technicznych w dyscyplinie elektrotechnika na Wydziale Elektrycznym Politechniki Częstochowskiej, odpowiednio w 1997 r. i 2003 r., a stopień doktora habilitowanego z informatyki na Politechnice Koszyckiej w 2013 r. Obecnie pracuje na stanowisku profesora uczelni na Wydziale Elektrycznym Politechniki Częstochowskiej. Jego zainteresowania badawcze i dydaktyczne dotyczą metrologii, pomiarów i oprzyrządowania, badań nieniszczących, teorii i zastosowania termografii w podczerwieni, systemów wbudowanych, elektroniki, przetwarzania i rozpoznawania obrazów. Jest autorem bądź współautorem kilku monografii naukowych, rozdziałów książek, podręcznika akademickiego, patentu i ponad 60 artykułów w czasopismach i materiałach konferencyjnych. Jest recenzentem w wielu renomowanych czasopismach wydawnictw IEEE, Elsevier, IOP i innych. Udziela się w pracach Komisji Metrologii Polskiej Akademii Nauk Oddział w Katowicach. Zajmuje się organizacją cyklicznych krajowych konferencji metrologicznych. Jest redaktorem działu metrologii kwartalnika PAN International Journal of Electronics and Telecommunications..


## prof. dr hab. inż. Waldemar Andrzej Minkina

waldemar.minkina@pcz.pl

**ORCID: 0000-0002-3153-270X**


Studia na Wydziale Elektrycznym Politechniki Częstochowskiej ukończył w 1977 r., doktorat – 1983 r., habilitacja – 1995 r., tytuł profesora nauk technicznych w dyscyplinie elektrotechnika na wniosek Wydziału Elektrotechniki, Automatyki, Informatyki i Elektroniki Akademii Gorniczo-Hutniczej w Krakowie w 2006 r. Specjalizuje się w problematyce dotyczącej szeroko rozumianej termometrii, termografii komputerowej, pomiarów cieplnych, inżynierii komputerowej oraz teorii sterowania. Jest autorem i współautorem 7 monografii z metrologii, 180 publikacji (w tym 45 w czasopismach z tzw. „listy filadelfijskiej" oraz wydawanych przez Polską Akademię Nauk i Akademie Nauk innych krajów) oraz 14 patentów. Był wykonawcą i współwykonawcą 16 prac zleconych dla przemysłu regionu częstochowskiego, które w większości zostały wdrożone. Za swoją działalność dwa razy otrzymał zespołową Nagrodę Ministra oraz około 20 nagród indywidualnych i zespołowych J.M. Rektora Politechniki Częstochowskiej. Obecnie pełni funkcję Kierownika Studium Doktoranckiego na Wydziale Elektrycznym PCz.