IV. KLASYFIKACJA JĘZYKÓW PROGRAMOWANIA

Do klasyfikowania języków programowania używać można wielu kryteriów. Niektóre z nich podano w niniejszym opracowaniu. I tak można te języki klasyfikować na języki:

1. wg techniki tłumaczenia -
   a/ interpretujące (interpretatory, interpretery, ang. interpreters),
   b/ kompilujące (kompilatory, kompilery, ang. compilers),
2. wg poziomu programowania -
   a/ wewnętrzne (maszynowe),
   b/ mnemoniczne operujące na adresach cyfrowych i stosujące symbole literowe do oznaczania kodu operacji,
   c/ mnemoniczne pełne (symboliczne, ang. assembly languages) posiadające symbole literowe operacji i argumentów,
   d/ symboliczne z makrorozkazami (o współczynniku tłumaczenia np. 4:1),
   e/ wyższego rzędu, tj. nie zorientowane maszynowo.
3. wg funkcji -
   a/ do przetwarzania informacji,
   b/ do obsługi przetwarzania (systemy operacyjne),
   c/ do pisania translatorów,
   d/ do opisywania języków (metajęzyki),
   e/ do innych celów,

(Powyższe grupy klasyfikacyjne niekiedy zachodzą na siebie. W szczególności dotyczy to grup a, c, d).

4. wg pojęcia proceduralności -charakteryzujące się tym, że programista podaje kolejność procedur czy rozkazów

(do tego typu należą więc prawie wszystkie języki),

a/ języki nieproceduralne tj. takie, w których programista podaje jedynie specyfikacje wejścia-wyjścia oraz wzory obliczeniowe, bez precyzowania sekwencji obliczeń (przykładem takiego języka może być REPORT PROGRAM GENERATOR).

Ze względu na ograniczoną objętość niniejszej pracy nie możemy omówić szczegółowo poszczególnych układów klasyfikacyjnych. Uwagę skoncentrujemy na językach do przetwarzania informacji.

ad 1. INTERPRETER charakteryzuje się tym, że natychmiast po dokonaniu tłumaczenia najmniejszej jednostki znaczeniowej programu (instrukcji) jest ona wykonywana, czyli otrzymujemy wynik. Jest to niewątpliwie zaleta z punktu widzenia użytkownika, wada zaś polega na tym, że trzeba dokonywać wielokrotnego tłumaczenia tych samych wyrażeń, np. przy obliczeniach cyklicznych.

KOMPILER tłumaczy najpierw cały program, a później dopiero po tzw. kompozycji (konsolidacji) program staje się gotowy do realizacji. Wprowadzenie zmian do programu pociąga za sobą konieczność rekompilacji.

Systemy interpretacyjne stosowane są z reguły w maszynach z małą pamięcią i dużą szybkością obliczeń.

Istnieją języki łączące w sobie zarówno cechy interpretatora jak i kompilatora (np. QUIKTRAN). Większość języków pracuje w systemach kompilacji. Jako przykład specjalnego systemu interpretacyjnego wymienić można systemy operacyjne.

Do systemu interpretacyjnego należą m.in. języki: MUMPS (MGH Utility Multi-Programming System) opracowany

na maszynę PDP-9 pod kątem wykorzystania wieloprogramowości oraz GOTRAN, stanowiący wersję FORTRANu na IBM 1620. Język MUMPS wykorzystuje zaletę interpretatora polegającą na tym, że programy są krótkie, i w związku z tym nawet przy niedużej maszynie można w pamięci przechowywać równocześnie kilka programów. Pozwala to na efektywne wykorzystanie właściwości podziału czasu (time-sharing) bez pośrednictwa pamięci dyskowej. Ponadto ułatwione jest uruchamianie programów, ponieważ istnieje możliwość bieżącej modyfikacji bez przebiegów rekompilacyjnych.

ad 2. Niewątpliwie podstawowym kryterium podziału języków jest dla programisty poziom programowania. Pisanie programów w języku wewnętrznym w maszynach III generacji (typu IBM seria 360, System 4, Seria 1900) nie jest (względnie prawie nie jest) stosowane. Wpływa na to kilka czynników (niezależnie od powyżej omówionych zalet języków wyższego rzędu):

a/ przydział adresów rzeczywistych w warunkach pracy wieloprogramowej realizowany jest nie przez programistę, lecz automatycznie przez system operacyjny,

b/ obsługa różnorodnych urządzeń wejścia-wyjścia wymaga zastosowania wielu procedur specjalnych i rozkazów dostosowanych do współpracy z systemem operacyjnym, zaprogramowanie których w programach użytkowych ("zastosowaniowych") za pomocą cyfrowych (wewnętrznych) instrukcji nastręczałoby poważne trudności.

c/ ponieważ łatwo jest o pomyłkę adresów i kodów operacji wyrażanych cyfrowo, oraz trudno je wykryć kontrolą formalną, uruchomienie złożonych programów trwałoby bar-

dzo długo, co oczywiście byłoby kosztowne (koszt pracy maszyny i programisty),

d/ z powodu długiego szkolenia i dużej pracochłonności programowania, byłoby trudno w krótkim czasie obciążyć maszynę w dostatecznym stopniu.

Stąd też zamieniono adresy i kody operacji symbolami łatwymi do zapamiętania (nazwami mnemonicznymi) oraz wprowadzono tzw. makrorozkazy zwalniające programistę od pisania typowych procedur.

Struktura instrukcji języka symbolicznego (poza makrorozkazami) zbliżona jest do struktury rozkazu języka wewnętrznego:

Etykieta (adres) instrukcji bloku	nazwa operacji	nazwy argumentów (operandów) lub adresy względne	nr indeksu (B-rejestru)

Jeśli chodzi o nazwy (symbole) operandów, to występuje ich tyle, ile jest adresów w rozkazie wewnętrznym. Programista posiada dostęp do poszczególnych rejestrów. Są to cechy szczególne, nie występujące w językach wyższego rzędu. Jedynie makrorozkazy traktować można jako swoisty rodzaj wstawek języka wyższego rzędu (jak wiadomo, języki wyższego rzędu budowane są z makrorozkazów).

Oto przykłady języków mnemonicznych:

a/ język mnemoniczny niepełny - City and Guilds Mnemonic Code:
ADD 302,5 oznacza - dodaj zawartość komórki o adresie (302 plus zawartość B rejestru Nr 5),

b/ mnemoniczny pełny (plus makrorozkazy)

- SAS (System Adresów Symbolicznych dla ZAM-41):
  SKO G8 - skocz do procedury z etykietą G8,
  PISZ, CZYTAJ - makrorozkazy (występują one również
  w języku wyższego rzędu SAKO)
- Assembler IBM 360:

SAMPLE START X, Ø	start programu od adresu wzgl. Ø,
START BALR 9, Ø	załad. rej. B zawartością adres. Ø,
USING =, 9	zdefiniuj rej. 9 jako rejestr bazowy,

- USERCODE dla Systemu 4:

MP WORK (6) PRICE (2)	mnóż WORK przez PRICE, wynik jest przechowywany w WORK,
ED DEST (11), VALUE	pole DEST jest redagowane w ten sposób, że nieznaczące zera są zamieniane przez znaki "spacje" lub "blank" względnie gwiazdkę,

- OPEN nazwa zbioru, nazwa zbioru ....., nazwa zbioru (do 16)
  jest to przykład makrorozkazu otwarcia zbioru powodującego sprawdzenie metryki zbioru, itp. (lub zapisanie),
  GET makrorozkaz przywołania rekordu ze zbioru wejściowego.

ad 3. Języki do przetwarzania informacji można podzielić na:
  a/ uniwersalne, np. PL/I,
  b/ specjalizowane (problemowe), np. COBOL, FORTRAN, ALGOL,
  c/ wąskospecjalizowane, np. języki do:
  - symulacji,
  - przetwarzamia struktur listowych (napisów, symboli),
  - redagowania wydawnictw graficznych i obsługi display'ów,
  - sterowania obrabiarkami,

- tłumaczenia tablic decyzyjnych itp.

Jeśli chodzi o języki do przetwarzania struktur listowych, to istnieje ich na świecie kilkadziesiąt, przy czym do najpopularniejszych należą: LISP, COMIT, SNOBOL. W Polsce opracowano języki LOGOL i EOL. Niektóre właściwości tych języków mogą być z powodzeniem wykorzystane do translacji z języka na język.

Specjalną grupę tworzą języki do operowania na formułach FORMAC (umożliwiają rozwijanie wielomianów, ich dzielenie): ALPAK, FORMULA, ALGOL, SAINT.

Do popularnych kwestii należy porównywanie języków do obliczeń numerycznych (a więc ALGOLu, FORTRANu) z językiem do przetwarzania danych ekonomiczno-administracyjnych (czyli z COBOLem).

Języki pierwszej grupy stosują wyłącznie zapis algebraiczny do oznaczania operacji arytmetycznych, podczas gdy COBOL oprócz tego zapisu (stosowanego po instrukcji COMPUTE) zapewnia również instrukcje słowne w rodzaju DIVIDE, MULTIPLY, ADD, SUBTRACT.

Składnia języków jest całkowicie odmienna. W COBOLu wprowadzono rozdzielanie opisu danych od procedur, stosując dwa odrębne rozdziały: DATA DIVISION i PROCEDURE DIVISION.

Ponadto specjalny rozdział służy do opisu i rezerwacji urządzeń (ENVIRONMENT DIVISION). Najważniejsze znaczenie z punktu widzenia translatora posiada rozdział identyfikacji IDENTIFICATION DIVISION. Tego rodzaju podział funkcjonalny nie istnieje w językach do obliczeń numerycznych, nastawionych na konstruowanie procedur i podających jedynie podstawowe informacje na temat danych (deklaracje INTEGER, REAL, FORMAT, DIMENSION).

Opis danych stosowany w COBOLu pozwala na operowanie nazwami zbiorów (w instrukcjach OPEN, READ), rekordów, pól grupowych

i pól elementarnych, a więc wyraża złożoność "montażową" (strukturę) danych administracyjnych. ALGOL i FORTRAN operują nazwami jednostkowymi (niepodzielnymi), dając możność stosowania wielokrotności pól jednorodnych (indeksowanych). Tę samą właściwość posiada zresztą również COBOL.

Z punktu widzenia alokacji danych język COBOL zapewnia z reguły bardziej efektywny program (dzięki możliwościom "pakowania" danych). Natomiast generuje dłuższe programy dla takich samych zadań (np. w porównaniu z FORTRANem).

Teoretyczne wersje ALGOLu pomijają sprawę formalnego opisu przydziału urządzeń wejścia-wyjścia oraz tzw. edytowania (redagowania) danych wynikowych, pozostawiając to producentom.

Efektem tego jest zmniejszenie stopnia zamienności programów pisanych dla różnych maszyn.

Oto parę instrukcji wydawniczych firmy ICL dla Systemu 4-50:

WRITE / 30, FORMAT / '/ 'ND' / ' / ,X/, gdzie: 30 jest nr urządzenia tj. drukarki wierszowej, ND oznacza wydruk 2 cyfr, X nazwa pola),

PAGE /30,1/ oznacza zmianę strony,

NEWLINE /30,1/ " skok do nowego wiersza.

Są to instrukcje "firmowe" bez odpowiednika w innych wersjach. Ponadto ALGOL nie uwzględnia sprawy rozpoznania stałego i zmiennego przecinka, pozostawiając to maszynie.

W niektórych wersjach REAL oznacza liczbę zmiennoprzecinkową.

ALGOL dla Systemu 4-50 dysponuje bogatymi możliwościami wydawniczymi, porównywalnymi do podobnych cech COBOLu, mianowicie w deklaracji FORMAT można m.in. używać następujących

symboli:

D - cyfra,

N - drukowanie "najstarszej" cyfry jeśli jest znacząca (tj. nie równa zeru),

S - spacja,

. - kropka dziesiętna,

"+" "-" - znaki arytmetyczne.

Przykłady redagowania:

opis	pole przed zredagowaniem	pole po zredagowaniu
7S ≠ NDDD	12	␣␣␣␣␣␣␣ + ␣ ␣ 12
3D4S.5D	123.456789	123␣␣␣␣.45678

Również języki wzorowane na ALGOLu wprowadzają czasem własne środki opisu danych, jak np. ALGEM (ALGOritmiczeskij jazyk dla programmirowania Ekonomiczeskich i Matimaticzeskich zadacz):

9 - cyfra w systemie dziesiętnym,

7 - cyfra w systemie ósemkowym,

1 - cyfra w systemie dwójkowym,

C - dowolny znak,

A - litera rosyjskiego alfabetu,

L - litera łacińskiego alfabetu.

. - kropka dziesiętna w postaci jawnej (dziurkowana),

Δ - kropka dziesiętna "założona",

Z - zamiana zer nieznaczących przez spację

± - znaki arytmetyczne.

Język ALGOL posiada dogodne właściwości operowania na zbiorach jednorodnych danych, czyli masywach (array). Przynajmniej teoretycznie nie ma tutaj ograniczeń co do ilości (głębokości) indeksów i możliwy jest dynamiczny podział pamięci (granice zbiorów mogą być wzorami arytmetycznymi).

Firmowe wersje ALGOLu wychodzą czasem dalej niż wersje formalne (publikowane jako REPORT ALGOL). Dotyczy to nie tylko wejścia i wyjścia, gdzie m.in. chodzi o powiązania z systemami operacyjnymi, lecz również o możliwość przywołania segmentów napisanych w języku symbolicznym (np. ALGOL 4-50 przywołuje segmenty napisane w USERCODE).

Język COBOL jest szczególnie przydatny w maszynach II generacji bez systemu operacyjnego, a to z tego względu, że operuje odpowiednim aparatem formalnym do opisu konfiguracji maszyny niezbędnej do wykonania zadań, umożliwia automatyczne sprawdzanie (i zapisywanie) metryk oraz deklaruje przydział urządzeń we-wy dla poszczególnych zbiorów.

Języki do obliczeń numerycznych są prostsze, a więc i łatwiejsze do wyuczenia się. Opracowuje się do nich krótsze translatory i w związku z powyższym można ich używać nawet w maszynach z małą pamięcią operacyjną (8 - 16 K).

Wdrożenie pierwszych kompilatorów na małych maszynach napotykało na spore trudności. Wyrazem tego były przypadki, kiedy do skompilowania programów trzeba było ponad 40 przebiegów (A - 1).

Mimo sztywności programu (obowiązkowe sekcje, rozdziały) i tłumaczenia go techniką kompilacyjną, język COBOL jest niekiedy stosowany w systemach abonenckich.

Wg danych pewnej ankiety (S-4, 1969 r.), którą objęto 497 użytkowników w/w systemów, struktura używanych przez nich języków programowania była następująca:

FORTRAN 39,6%,
BASIC 31%,
COBOL 7,6%,
PL/I 7,0%,
ALGOL 2,8%.

Jeśli chodzi o język BASIC (Beginner's All-purpose Symbolic Instruction Code), to opracowano go w Dartmouth College (w okresie 1964-1965) i wdrożono na maszynie GE 225. Pomyślany był, jako język do wstępnej nauki programowania przed przystąpieniem do ALGOLu lub FORTRANu. Posiada proste konstrukcje formalne, będąc w ten sposób bardzo łatwy do wyuczenia (w zasadzie programuje się już po pierwszym dniu nauki). Uruchamianie programów jest ułatwione. W dowolnej chwili można wprowadzać poprawki, podając jedynie nr sekwencji (zdania) i nowe jej brzmienie. Jako inne języki konwersacyjne wymienić można AMTRAN, JOSS, CAL, QUIKTRAN. Częściowo słusznym zarzutem (z dokumentacyjnego punktu widzenia można mu się przeciwstawić) w stosunku do COBOLu jest zbytnia obfitość części opisowych, jakiej musi używać programista. Szczególnie odnosi się to do opisu danych, wyrażania operacji arytmetycznych (jeśli brak instrukcji COMPUTE). Wada ta daje się częściowo usunąć, jeśli stosowane są instrukcje COPY (do ściągania z biblioteki zarówno opisu danych jak i procedur, wspólnych dla kilku programów). oraz dozwolone jest użycie takich skrótów, jak PIC do oznaczenia PICTURE, COMP zamiast COMPUTATIONAL itp.

Znane są specjalne wersje COBOLu, używające jedynie minimalnych oznaczeń. Na przykład wersja COAX stosuje jednoliterowe

oznaczenia :

- F' oznacza FILLER /w DATA DIVISION/ lub
  FROM / w PROCEDURE DIVISION/
- A' ADD
- G' GREATER,

itp.

Inne usiłowania polegają na zastosowaniu specjalnych formularzy z gotowymi nadrukami (np. RAPIDWRITE).

Werbalność COBOLu związana jest niewątpliwie z celem, jaki był stawiany przed tym językiem, kiedy chodziło o zbliżenie specyfiki programowania do specyfiki problemu (a więc do ludzi pracujących w określonym zawodzie).

Ponadto okazało się, że z pozoru proste tematy administracyjne są bardzo pracochłonne w programowaniu i należało stworzyć programiście takie narzędzie pracy, które pozwala szybko uruchamiać bardzo duże programy(to znaczy łatwo je analizować i szybko odnajdywać błędy) i będzie równocześnie stanowić dobrą dokumentację do kontynuacji pracy przez innych programistów.

COBOL bardziej uwzględnia wymogi nowoczesnej technologii przetwarzania, wykorzystując np. walor pracy równoczesnej (overlapping), dzięki możności podwójnego buforowania każdego wejścia i wyjścia oraz obowiązkowemu wprowadzenia odrębnych obszarów wejściowych i wyjściowych.

Pod względem czasu przetwarzania zadań angażujących duże zbiory i złożone rekordy język COBOL zdecydowanie wyprzedza języki do obliczeń numerycznych (w tym również FORTRAN posiadający jakie takie możliwości opisu wprowadzanych danych - za pomocą deklaracji FORMAT).

Oto wyniki pewnych testów (J - 1) przeprowadzonych na maszynie IBM 7094 w połowie lat 60-tych:

zadanie	COBOL	FORTRAN II
1. Czytanie zbiorów i sprawdzanie jakości pól rekordów		
a/ wielkość bloku 10 rek.	2,3 min.	15 min.
b/ wielkość bloku 1 rek.	4,8	10,2
2. Kopiowanie zbioru		
rozmiar bloku 10 rek.	2,8	13,8

Można powiedzieć, że wraz ze wzrostem pojemności pamięci i szybkości działania maszyn wzrasta efektywność kompilatorów COBOLu. Tam, gdzie tego nie udaje się osiągnąć w trakcie translacji, stosowane są specjalne optymalizatory (COBOL optymiser), które przerabiają wersję otrzymaną w wyniku tłumaczenia. Przykładem takiego optymalizatora jest COBOL OPTIMISER opracowany w Capex Corp. of Phoenix Arizona (X - 2).

Nie można mówić o efektywności kompilatora w sensie ogólnym. Uważa się, że dobre kompilatory COBOLu opracowuje firma Honeywell, o czym świadczy fakt, że udział wdrożonych kompilatorów tej firmy jest znacznie wyższy od udziału w sprzedaży maszyn.

Wydaje się, że na temat przydatności COBOLu przede wszystkim powinni wypowiadać się użytkownicy. Otóż wg ankiety (X - 3) przeprowadzonej w USA wśród 724 użytkowników COBOLu, 560 z nich

(ok. 80%) potwierdziło pełną przydatność tego języka, przy czym 65% z tej grupy oświadczyło, że zakres stosowania tego języka stale się u nich zwiększa.

Prowadząca ankietowanie Auerbach Corp. podała wyniki mówiące, że użycie COBOLu jest dwukrotnie wyższe w takich zastosowaniach, jak ewidencja zapasów, rozliczenia zakupów, sprzedaży itp. - zaś język symboliczny dorównywał COBOLowi jedynie w przypadkach przetwarzania wyrywkowego (random access job.).
Największymi zwolennikami COBOLu były najczęściej ośrodki małe, zatrudniające do kilkunastu programistów.[1/]
Reasumując zalety COBOLu stwierdzić można, że jest to język stosunkowo łatwy do wyuczenia i stosowania (w porównaniu z językami symbolicznymi) oraz przedstawiający duże walory dokumentacyjne. Dzięki zastosowaniu zwrotów PERFORM ...., DEPENDING ON, itp. upraszcza się organizacja programu, zaś w czasie testowania pomocne mogą być procedury TRACE, EXIDIT itp.

Wadami COBOLu, w porównaniu z językami symbolicznymi, są:
a/ większa, np. o 30%, zajętość pamięci operacyjnej,
b/ dłuższy, np. o 10%, czas przetwarzania (T - 2),
zaś zaletami: kilkakrotne zmniejszenie pracochłonności programowania i czasu uruchamiania programów.

Skoro mówimy o językach do przetwarzania danych, nie możemy pominąć arcyciekawej próby (1959, USA) stworzenia "algebry informacji" w postaci języka LSG /Language Structure Group/.

1/ Na poparcie danych literaturowych mogę od siebie dodać, że w czasie mego 7-miesięcznego pobytu w Wielkiej Brytanii spotkałem ośrodki obliczeniowe, w których posługiwano się prawie wyłącznie językiem COBOL.

Znane są próby zastosowania tego języka do obliczania list płac.

Analiza porównawcza języków programowania jest utrudniona, ponieważ stale ulegają one zmianom, zarówno w wersjach formalnych, jak i firmowych. Obserwuje się m.in. wzajemne zapożyczenia, istnieją grupy języków wzorowane na FORTRANie, ALGOLu i COBOLu, względnie czerpiące ze wszystkich tych języków na raz.

Przykładem wpływu FORTRANu na ALGOL może być deklaracja FORMAT używana w niekórych wersjach ALGOLu po instrukcji WRITE (to też niestandardowa instrukcja) oraz instrukcja READ z numerem urządzenia jako parametrem.

Niektóre języki (czy też raczej pakiety programowe) posługują się wprost aparatem formalnym innego języka, jak to jest w przypadku BASEBALL (system pytanie-odpowiedź), używającego zwrotów języka IPL-V do przetwarzania struktur listowych.

Wyrazem pewnego podsumowania dorobku programowania wydaje się być uniwersalny język PL/I. Stanowi on zjawisko ze wszech miar godne uwagi. Nie stał się jeszcze jednak językiem powszechnego stosowania (w 1968 roku tylko 1% komputerów miało opracowany translator tego języka - wg R - 3).

W celu zbadania efektywności tego języka w zastosowaniach ekonomiczno-administracyjnych przeprowadzono testy porównawcze z COBOLem na przykładzie konkretnych programów m.in. dotyczących list płac (R - 3). Porównanie to nie podważyło pozycji COBOLu. Oto wyniki:

a/ uruchamianie programów PL/I trwało znacznie (dwukrotnie) dłużej, mimo krótszych programów zewnętrznych,

b/ czas pisania programów PL/I był tylko nieznacznie krótszy;

c/ czas przetwarzania był w przypadku COBOLu znacznie krótszy (dla dwóch programów wynosił 2/3 i 1/3 czasu PL/I);

d/ uczenie się języka PL/I jest znacznie trudniejsze, natomiast później użycie aparatu formalnego wydaje się być łatwiejsze. Jest to więc niejako język przeznaczony dla zawodowych programistów, specjalizujących się w programowaniu PL/I (wtedy rekompensują się nakłady na szkolenie);

e/ PL/I zapewnia lepsze możliwości kontroli logicznej i operowania na danych (włącznie z manipulacją na bitach, nie zawsze stosowaną w COBOLu). Natomiast COBOL posiada lepsze procedury czytania i pisania (włącznie z takim ułatwieniem wydawniczym, jakim jest REPORT-WRITER);

f/ łatwiejsza jest korekta czy też modyfikacja programów PL/I.

Tyle mówią praktyczne konfrontacje. Natomiast pod względem poziomu programowania, język PL/I przewyższa pozostałe języki. Jest próbą opracowania języka adekwatnego do hardware'u i systemu operacyjnego maszyn III generacji.

Język PL/I opracowano, starając się uwzględnić osiągnięcia istniejących języków, a w szczególności FORTRANu, ALGOLu i COBOLu.

Notacja języka PL/I jest stosunkowo zwięzła i w tym względzie PL/I opiera się raczej na tradycjach FORTRANu i ALGOLu, niż COBOLu. Język ten uważany jest za szczytowe osiągnięcie języków proceduralnych. Ponieważ ma być samowystarczalny, nie przewiduje on wstawek z innych języków (przynajmniej nie przewidywały tego pierwsze wersje).

Specyficzną cechą PL/I jest powiązanie z systemem operacyjnym i hardware'm. Wyraża się to przede wszystkim w organizowaniu pracy równoległej, zarówno w przetwarzaniu wielomaszynowym jak i w maszynie wieloprogramowej. Przykładem tego rodzaju pracy jest tzw. wieloproceduralność (lub podprogramowanie), polegająca na tym, że równolegle wykonuje się kilka procedur tego samego programu, przy czym zorganizowane może to być asynchronicznie lub w systemie przerywań.

Asynchroniczność osiągana jest m.in. poprzez stosowanie takich instrukcji, jak: wstrzymanie procedury WAIT, przerwanie procedury na określoną ilość jednostek czasu DELAY, badanie ukończenia COMPLETE itp.

Z punktu widzenia użytkownika specjalne znaczenie posiadają dwie cechy PL/I: modułowość i zasada domyślności.
Modułowość sprawia, że nie trzeba znać języka w całości, lecz stosuje się jego podzbiory stosownie do poziomu programisty i złożoności zastosowania.

Dzięki zasadzie domyślności, w przypadku wykrycia braku deklaracji translator nie sygnalizuje błędu, lecz dobiera deklarację najbardziej prawdopodobną w danym przypadku. Na przykład, jeśli przy zmiennej EXTERNAL nie ma deklaracji sposobu przydziału pamięci, translator przyjmuje STATIC.

Bardzo zwięzły w porównaniu z COBOLem jest sposób opisu struktury danych (połączony od razu z indeksowaniem):

```
DECLARE 1A/2/, 2B, 2C   oznacza:poziom 1  A/1/
                                          2 B
                                          2 C
                                        1 A/2/
                                          2 B
                                          2 C
```

W PL/I rozróżnia się dwa sposoby przesyłania danych:
a/ zorientowany na tzw. strumień,
b/ zorientowany na rekord (zapis, dokument).
Do wejścia-wyjścia strumienia używa się instrukcji GET i PUT, zaś w drugim przypadku READ i WRITE. Przy strumieniowym przesyłaniu dane są rozpatrywane jako ciągły strumień elementów w postaci znakowej. W rekordowym przesyłaniu jednostką przesyłową jest rekord, przy czym jego elementy mogą być zakodowane w różnych formach.

Głównymi pojęciami opisu danych są maszyny i struktury. Przyrównując je do tradycyjnych pojęć, struktura odpowiada rekordowi, zbiorowi jako złożona struktura), zaś masyw - tabeli jednorodnych danych, np. macierzy.

Sposób kompilacji programów PL/I różni się znacznie od dotychczasowych rozwiązań. Mianowicie obejmuje również elementy przetwarzania (np. obliczanie wspólnych lub jednorazowych wyrażeń) oraz umożliwia programiście włączenie się czynne do procesu kompilacji.

Języki wąskospecjalizowane

Języki te służą do programowania takich specjalnych zastosowań, jak:
a/ sterowanie obrabiarkami (machine tool control) np. APT,
b/ konstrukcje logiczne np. LOTIS, APL,
c/ budownictwo cywilne np. COGO, STRESS,
d/ pisanie translatorów np. CLIP, COGENT,
e/ symulacja cyfrowa np. GPSS, SIMSCRIPT, SOL,
f/ systemy pytanie-odpowiedź np. COLINGO, BASEBALL, QUERY, ADAM, DEACON,

g/ wyjścia graficzne i ekranowe np. GRAF, PENCIL, DOCUS,

h/ tłumaczenie tablic decyzyjnych.

Przeznaczeniem języków wąskospecjalizowanych jest obsługa wybranego rodzaju zastosowania, w którym są łatwiejsze w użyciu i efektywniejsze od innych. Wynika to z wyposażenia tych języków w specjalistyczne instrukcje, jak np.

a/ w APT : SPINDL/OFF (turn off spindle - obtoczyć wał)

TL RGT, GO PGT/BASE (with the tool on the right go right along the line BASE)

b/ COLINGO: UPDATE, CHANGE, EXECUTE, ALLOCATE, ANALYZE.

Skoro wspomnieliśmy o języku APT, warto dodać, że opracowany został w MIT (Massachusetts Institute of Technology) w latach 1952 - 1956. Ulepszone wersje to:

1958 APT II, 1961 APT III.

Jeśli chodzi o język do symulacji, to jednym z pierwszych języków tego typu był język DYNAMO opracowany w MIT w 1959 roku dla IBM 704. Najbardziej znanymi językami w zakresie symulacji są GPSS, SIMSCRIPT i SOL.

Opis GPSS (General Purpose Simulation System) po raz pierwszy opublikowano w 1961 roku. SIMSCRIPT (Simulation Oriented Language) został opracowany w RAND CORP. w 1960 roku.

SOL (Simulation Oriented Language) stanowi próbę połączenia dwóch różnych systemów językowych, a mianowicie systemu "blokowego" (GPSS) i "zdaniowego" (SIMSCRIPT).

Języki do pisania translatorów można podzielić na dwie grupy:

1. oparte o zasady zwykłych języków programowania, np. CLIP (Compiler Language for Information Processing) oparty został na ALGOLu 58, lecz posiada istotne uzupełnienia w zakresie manipulacji na danych (np. podanie rozmiaru nieindeksowanych danych) oraz instrukcje wejścia-wyjścia,
2. do tzw. tablicowego tłumaczenia składni (syntax directed table-driven compilation), np. COGENT (Compiler and GENeralized Translator) opracowany dla CDC 3600.

W celu pokazania specyfiki języków do pisania translatorów przytoczymy przykład instrukcji w języku TMG:
INTEGER...ZERO-MARKS DIGIT *INSTALL co oznacza: pomiń nieznaczące zera, zaznacz początek łańcucha i znajdź co najmniej jedną cyfrę, a następnie wyspacjuj wszystkie następne cyfry oraz umieść "symbol" do tabeli znaków i drzewa (grafu) wyjścia.

Generatory programów

Przez generację rozumiemy automatyczne tworzenie typowych (powtarzalnych) sekwencji operacji w oparciu o "skrótowe" dane (rodzaj sekwencji, parametry) dostarczone przez programistę. Ze względu na radykalne zmniejszanie pracochłonności programowania, zastosowanie generatorów wydaje się mieć duże perspektywy.

Wyróżnić można trzy podstawowe poziomy generacji:

a/ makrogenerator,

b/ generator programu standardowego,

c/ generator-język programowania.

a/ Makrogenerator

Generuje rozkazy elementarne na podstawie makrorozkazu i deklaracji (opisu danych itp.) umieszczonych w programie zewnę-

trznym.[1/] Makrogeneratory są szczególnie przydatne do wyrażania operacji o wielu możliwych kombinacjach (np. typów operacji dodawania, zależnych od charakteru danych, żądanych zaokrągleń, kontroli formatu wyniku itp.). Kody tych operacji są <u>każdorazowo</u> generowane.

b/ <u>Generator programu "standardowego"</u>

Służy do generowania takich programów, jak np. SORT, REPORT, WRITER, które w klasycznej postaci posiadają sztywną formę programów standardowych. Zaletą metody jest wyeliminowanie ograniczeń programu standardowego i wyższa efektywność. W przypadku generacji SORT, generator na podstawie parametrów i zasadniczego szkieletu działania programu generuje program sortowania dopasowany do konkretnych wymogów (rozmiarów i typów rekordów, rodzaju i ilości kluczy sortowania itp.) oraz dostępnej konfiguracji. Jeden z pierwszych generatorów SORT został napisany przez F.Holbersona jeszcze w pierwszej połowie lat 50-tych (S - 1).

REPORT-WRITER służy do wybierania danych ze zbiorów oraz organizacji ich wydruku. Istnieją również generatory do aktualizacji zbiorów, generowania kompilatorów (tzw. compiler-compiler, compiler generator) itp.

c/ <u>Generator - język programowania</u>

Jako przykład tego rodzaju (nieproceduralność !) języka programowania wymienić można RPG-REPORT PROGRAM GENERATOR, opracowany m.in. dla maszyn IBM 1401, seria 360, UNIVAC.

1/ Inną drogą realizacji makrorozkazu - z pominięciem generacji - jest przywołanie gotowego podprogramu.

RPG umożliwia proste zaprogramowanie nawet złożonego problemu z dziedziny przetwarzania danych administracyjno-ekonomicznych (obejmującego szereg operacji obliczeniowych, wydruki wg różnych stopni kontroli itp.).

Programista posługuje się standardowymi arkuszami programowania o nadrukach odrębnych dla:

1/ opisu zbiorów (rodzaje),

2/ opisu struktury rekordów wejściowych,

3/ opisu danych wyjściowych na drukarkę wierszową (pola, tytuły, sumy wg stopni kontroli, znaki wydawnicze),

4/ opisu obliczeń (rodzaje operacji, sposób kontroli obliczeń).

Zwolennicy RPG twierdzą (R - 6), że 80 - 90% programów przetwarzania danych może być z powodzeniem napisane w tym języku. Optymiści (X - 4) oczekują, że w najbliższej przyszłości 75% wszystkich programów przetwarzania danych zostanie napisanych w RPG, podczas gdy tylko 15% przypadnie na COBOL a 10% na BAL i FORTRAN.

Systemy operacyjne (CONTROL PROGRAM, CONTROL LANGUAGE, CONTROL SOFTWARE, OPERATING SYSTEM)

Opracowanie systemów operacyjnych stanowi problem równie ważny jak rozwój "użytkowych" języków programowania. Chodzi przede wszystkim o zmniejszenie przestojów maszyny (poprzez zredukowanie do minimum czynności operatora, automatyczne przełączanie pracy maszyny z zadania na zadanie, harmonogramowanie pracy urządzeń) oraz stworzenie języka komunikacji operatora z maszyną. Wymogi systemów operacyjnych coraz bardziej rzutują na konstrukcję języków programowania.

System operacyjny jest niezbędny w maszynach wieloprogramowych, a znaczenie jego szczególnie wzrasta w wielomaszynowym przetwarzaniu. Dzięki niemu uzyskać można tzw. POLIMORFICZNOŚĆ, polegającą na tym, że zestaw komputerowy zmienia swą "postać" stosownie do problemów, poprzez zmianę połączeń pomiędzy elementami składowymi konfiguracji oraz wprowadzanie zmian strukturalnych (zmiana długości słowa, akceptowanie innej listy rozkazów itp).

Pierwsze publikacje na temat systemów operacyjnych pojawiły się w 1953 roku i dotyczyły systemów SOS (Share Operating System) i FMS (Fortran Monitor System) opracowanych dla maszyn IBM 709 i 704 (T - 2).

Firma Honneywell pierwszy swój system operacyjny opracowała w 1957 roku dla maszyny D-1000, zaś w 1960 roku stworzyła pakiet EXECUTIVE do kierowania pracą wieloprogramową (do 7 programów / X - 5 /.

Proste systemy operacyjne (zwane w IBM dyrygentami, w odróżnieniu od szerszego pojęcia systemu operacyjnego), składające się z kilkuset rozkazów, służyły głównie do wzywania standardowych podprogramów umieszczonych na taśmie bibliotecznej. Przykładem takiego dyrygenta jest MONITOR-70 dla maszyn IBM 7070/7074 (F-1).

Nieco większe systemy operacyjne sterowały również pracą urządzeń wejścia-wyjścia oraz realizowały kontakt operatora z maszyną (i odwrotnie). Systemy operacyjne maszyn III generacji kierują pracą wieloprogramową, przydzielają urządzenia wejścia-wyjścia, dokonują przydziału obszarów pamięci operacyjnej itp.

Przez pojęcie systemu operacyjnego firma IBM i firma Honeywell rozumieją cały kompleks oprogramowania sterującego. Przybliżeniem do tej koncepcji był system MONITOR MACDONALD dla maszyn IBM

709/7090, obejmujący swym zasięgiem kierowanie tłumaczeniami. Jedną z pierwszych wersji systemu operacyjnego IBM był SO dla IBM 1410/7010, zaś jej pełnym wcieleniem - oprogramowanie serii 360.

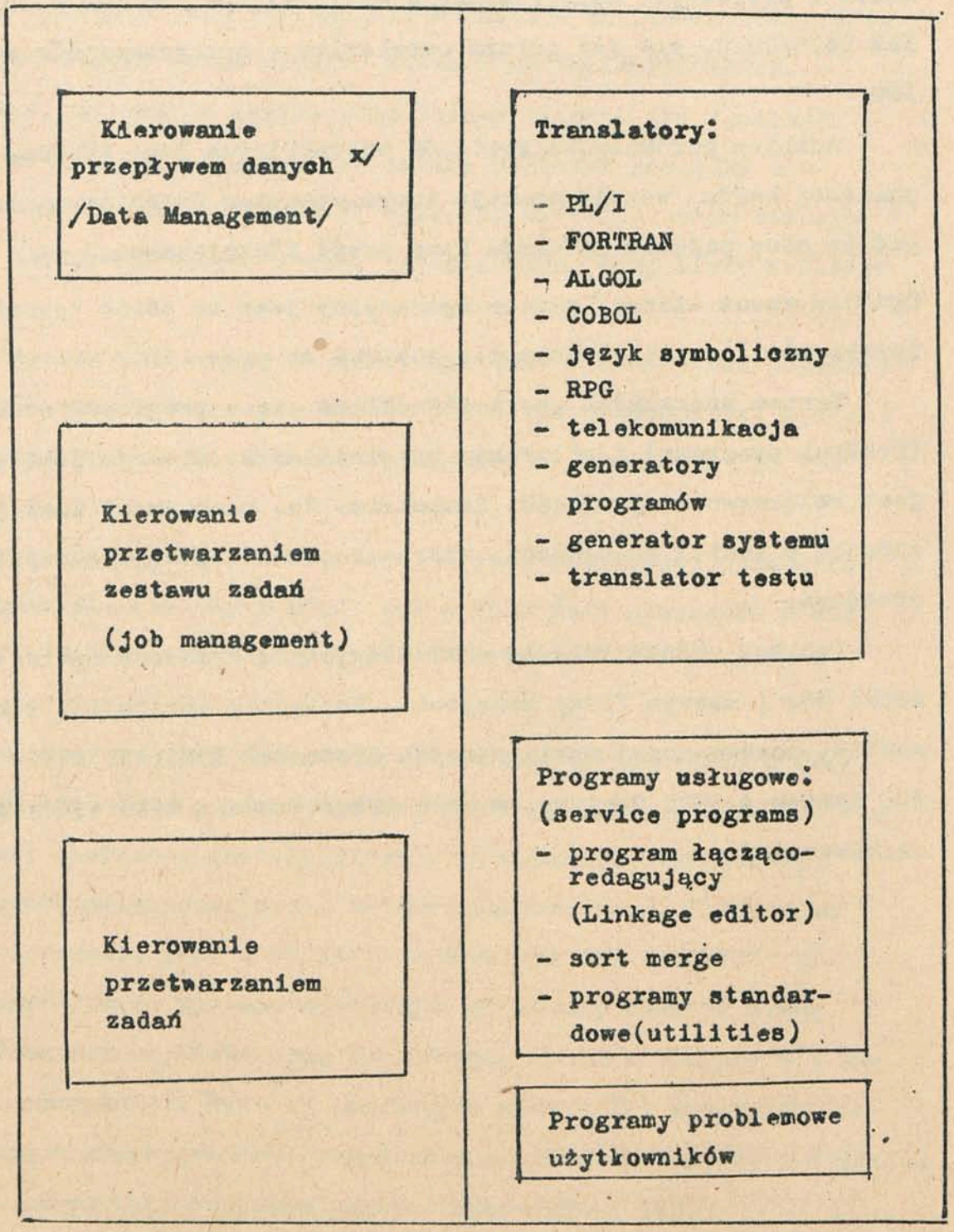
Analiza porównawcza systemów operacyjnych jest trudna, ponieważ każda wersja operuje innym aparatem pojęciowym, zaś pod te same pojęcia podkłada inną treść funkcjonalną.

Ogólnie rzecz biorąc, system operacyjny jest to zbiór "prawideł" (programów), czyniący komputer zdolnym do pracy.

System operacyjny serii 360 składa się z programów sterowania (Control programs) i programów przetwarzania. Głównym jego celem jest zwiększenie wydajności komputera. Np. Management task grupuje zadania w takiej kolejności, która zapewni lepsze wykorzystanie urządzeń.

Poniżej podamy zwięzłą charakterystykę systemów operacyjnych serii 360 i maszyn firmy Honeywell. Następnie spróbujemy dokonać analizy porównawczej następujących systemów: IBM, ICL seria 1900, ICL system 4, NCR Century, w celu zilustrowania istniejących rozbieżności.

Tabela 1. system operacyjny IBM 360



x/ W literaturze krajowej spotyka się też synonimiczne terminy angielskiego pojęcia: "data management", a mianowicie: "operowanie danymi", "sterowanie danymi" lub "sterowanie ruchem danych" (Przyp.Red.)

Tabela 2. System operacyjny Honeywell OS/200

I. S U P E R V I S O R

1. MONITOR STAŁY /Resident Monitor/
   - wprowadzanie programów,
   - kierowanie pracą wieloprogramową jedno lub dwustrumieniową,
   - kierowanie wejściem-wyjściem /alokacja kanałów przesyłowych i urządzeń,
   - kierowanie konwersją danych /karty-taśma magn., taśma magnet.-drukarka, taśma magn.-dziurkarka taśmy,
   - kierowanie komunikacją

2. MONITOR PRZEJSCIOWY /Transitional Monitor/
   - wczytywanie kart sterujących następnego programu i przekazanie parametrów do monitora stałego,
   - inicjowanie komunikacji,
   - wykonywanie dumpingu /PAO, dysku, TM/

3. KIEROWANIE ZBIORAMI BAZY DANYCH - składnik opcjonalny /data base file management/

II. K O M P O N E N T Y Z A L E Ż N E /Dependent Components/

1. PODSYSTEM KIEROWANIA DANYMI /dla zbiorów w papięci dyskowej/
   - kierowanie dostępem, założenie i aktualizacja rozmieszczenia zbiorów, zamiana ścieżek, załadowanie zbiorów, listowanie rozmieszczenia /map/
   - podprogramy wejścia-wyjścia: pakiety logiczny i fizyczny we-wy, itp.

2. PODSYSTEM JĘZYKÓW I PROGRAMÓW
   - translatory, archiwowanie programów źródłowych i wewnętrznych.

3. PROGRAMY STANDARDOWE

Tabela 3. Zestawienie porównawcze systemów operacyjnych

IBM: BOS, DOS, TOS	ICL: seria 1900	ICL: system 4	NCR: Century
1. IPL Loader /initiate program load./ program inicjujący wprowadzanie 2. SUPERVISOR -obsługa przerywań, - sterowanie wejściem-wyjściem, - sygnalizowanie błędów, - koordynacja przejścia od task control do job control 3. JOB CONTROL	1. EXECUTIVE - kontakt z operatorem, - ładowanie programów, - przydzielanie urządzeń, - organizacja pracy wieloprogramowej. 2a. AUTOMATYCZNY OPERATOR 2b. GEORGE /General Organisational Environment/ stanowi rozwinięcie funkcji EXECUTIVE GEORGE IV: - stronicowanie, - harmonogramowanie prac, - rejestracja wykorzystania jedn.centr. i urządzeń we-wy przez poszczególne programy GEORGE III: zawiera m.in. MOP /Multiplo On-line Programming Module/ umożliwiający pracę w wielodostępie dla 60 abonentów.	1. EXECUTIVE a/Job Control - przydział pamięci, - przydział urządzeń - wprowadzanie programu /Job input, Job scheduler Allocator Deallocator/ b/Supervisor - kontakt z operatorem, - system przerywania, - kronika błędów, - nadzór nad pracą wieloprogram. /S05J-6 strumieni, 14 programów/ 2. PROGRAM TRIALS SYSTEM 3. UTILITIES /program standard./	1. MONITOR - wprowadzanie programu, - kierowanie kolejnością pracy. 2. INPUT-OUTPUT EXECUTIVE - przydział pamięci, - przydział urządzeń wejścia-wyjścia, - kontrola błędów. 3. KIEROWANIE PRZEPŁYWAMI DANYCH /data traffic control/ - harmonogramowanie, - system przerywań.

Jak wynika z powyższego, rozbieżności dotyczą szczególnie takich pojęć, jak SUPERVISOR i EXECUTIVE, które w każdym przypadku znaczą właściwie co innego przy zachowaniu pewnych wspólnych funkcji.

Ogólnie rzecz biorąc system operacyjny spełnia następujące funkcje:

- wprowadzenie programów i kontrola ich realizacji,
- interpretacja rozkazów operatora i ich wykonanie,
- raportowanie odchyleń operatorowi (np. nadmiaru, przekroczenia limitu pamięci itp),
- interpretacja kart sterowania (CONTROL CARDS, zwane też Steering Cards),
- harmonogramowanie przebiegów,
- kontrola pracy urządzeń peryferyjnych,
- przydział urządzeń peryferyjnych do programów i zwalnianie tych urządzeń,
- zatrzymanie programu, o ile dalsza jego realizacja nie może być kontynuowana oraz powtórne uruchomienie po usunięciu przyczyny zatrzymania,
- kierowanie pracą wieloprogramową.

Złożoność systemu operacyjnego SO stale wzrasta np. SO dla maszyny IBM 360-91 zawiera ponad półtora miliona instrukcji. GEORGE III wraz z EXECUTIVE zajmuje około 12 tysięcy słów pamięci operacyjnej.

System operacyjny MASTER (CDC 3300) stosowany jest w maszynach wyposażonych w co najmniej 64 K PAO. Dzięki temu systemowi, uzyskano tak sprawną pracę wieloprogramową, że przepustowość maszyny zwiększyła się o 40% (X - 6).

System operacyjny SIPROS (SImultaneous PRocessing Operating System) dla CDC 6000 stanowi przykład kierowania bardzo złożoną konfiguracją 11 jednostek przetwarzania, w której występuje równoczesność: wykonywania programów i instrukcji, dostępu do poszczególnych modułów pamięci operacyjnej oraz wykonywana jest

tzw. rekonfiguracja (dodanie nowego składnika, usunięcie uszkodzonego).

V. GENEZA JĘZYKÓW POWSZECHNEGO ZASTOSOWANIA

Pisząc o językach powszechnego zastosowania mamy na myśli nie ich uniwersalność, lecz liczbę wdrożeń translatorów i uruchomionych programów.

Na pewno za takie języki można uważać ALGOL, FORTRAN i COBOL, zaś język PL/I (jako uniwersalny i nowoczesny) wydaje się mieć szanse zostania takim językiem.

Dużą przeszkodą w osiągnięciu powszechności języków programowania były zaściankowe interesy firm. Po pierwszym okresie przykrych doświadczeń specjaliści dochodzą do wniosku, że znaczenie języków programowania wykracza daleko poza rolę narzędzia walki konkurencyjnej i kwestia jakości (oraz powszechności) języków nie jest sprawą wewnętrzną jednej firmy. W ten sposób doszło do powstania języków ALGOL i COBOL, jako produktu współpracy przedstawicieli firm i naukowców.

1. Geneza ALGOLu

ALGOL (ALGOrythmic Language) został opracowany przez przedstawicieli Anglii, Danii, Francji, Holandii, NRF, Stanów Zjednoczonych i Szwajcarii. Wersję języka przygotowano na szeregu spotkań odbytych na terenie Europy i Stanów Zjednoczonych.

Największe zasługi wydają się mieć dwie organizacje: