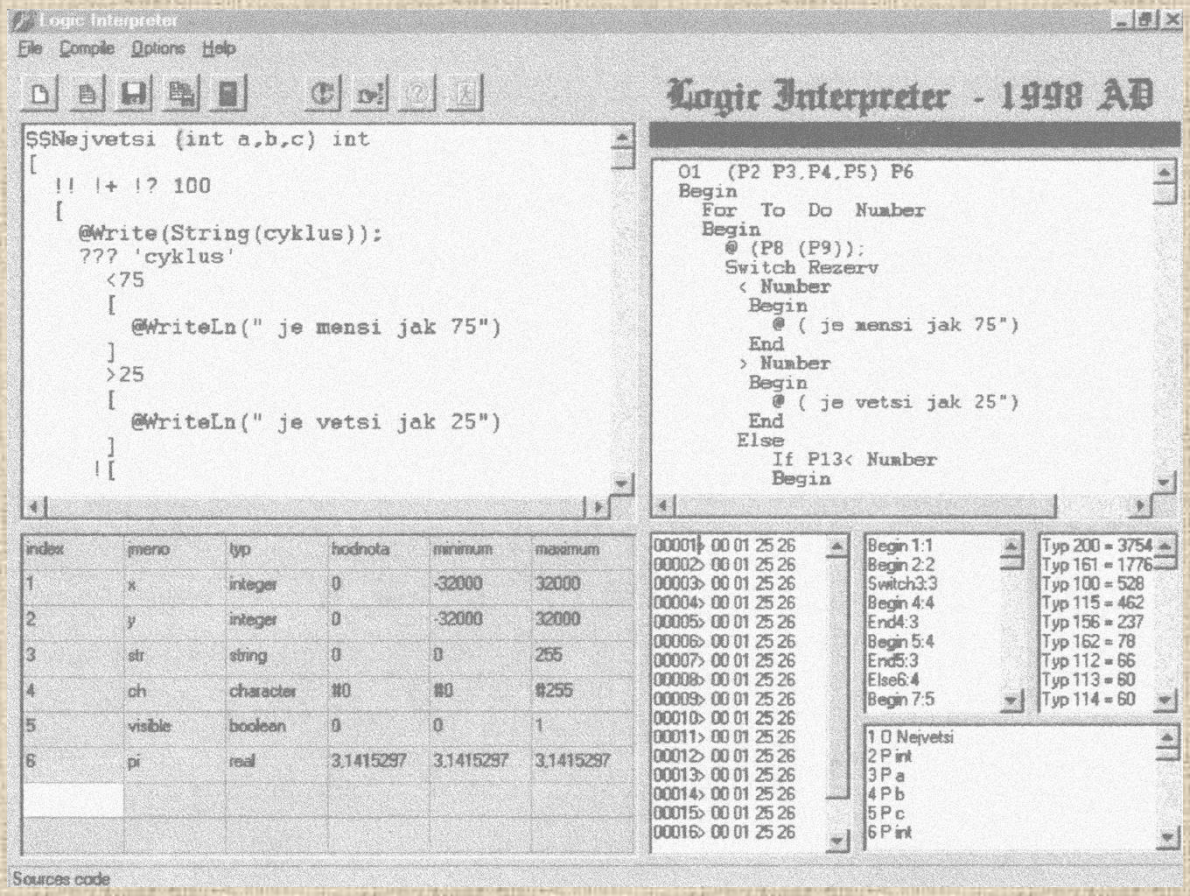


NÁVRH PROGRAMOVACÍHO JAZYKA WIRTH

(PREHISTORICKÁ) SEMINÁRNÍ PRÁCE K POCTĚ **NIKLAUSE WIRTHA**

Zadání

Navrhněte a definujte vyšší programovací jazyk umožňující strukturované programování. Jazyk musí umět definovat a pracovat se všemi základními datovými typy, indexovanými a strukturovanými proměnnými, obsahovat základní řídící struktury a příkazy. Jazyk nesmí být závislý na žádném přirozeném jazyce a počítači. Výrazové prostředky jazyka musí být maximálně stručné a čitelné

Původní text: **18. srpna 1998**
Revize: **16. září 2022**

Jiří (Vyšín) **Altior**
Vyšší odborná škola, Pardubice, Karla IV. č. 13

Anotace

Tato diplomová práce řeší problematiku spojenou s návrhem nového vyššího programovacího jazyka Wirth.

Na začátku je popsán vývoj programovacích jazyků a jejich charakteristické vlastnosti. Následuje teoretický a praktický postup při návrhu programovacího jazyka Wirth.

Na závěr jsou shrnuty nejdůležitější vlastnosti jazyka Wirth.

Annotation

This thesis tackles a problem of the project of new high-level programming language Wirth.

There is a principle of programming languages development and theirs characteristic properties at the beginning. The follows the theoretic and practic progress by the programming language Wirth developing.

There are the most important properties of the programming language Wirth at the end.

Zadání

Navrhněte a definujte vyšší programovací jazyk umožňující strukturované programování. Jazyk musí umět definovat a pracovat se všemi základními datovými typy, indexovanými a strukturovanými proměnnými, obsahovat základní řídící struktury a příkazy. Jazyk nesmí být závislý na žádném přirozeném jazyce a počítači. Výrazové prostředky jazyka musí být maximálně stručné a čitelné

POZNÁMKA K REVIDOVANÉMU VYDÁNÍ

Programovací jazyk Wirth je jenom studentskou hříčkou, hravou úvahou nad minimalisticky pojatou syntaxí strukturálního programovacího jazyka. Nejedná se ani o převratnou technologie, ani nic hodného zapamatování. Přesto jeho revidované vydání má svůj význam. Ukazuje, že i školní práce mohou být zábavné, invenční a mít svůj vtip, byť neposlouží pozdvihnutí národního hospodářství ani nezpůsobí evoluci v oboru. Studenti by si měli uchovat radost z překračování hranic, smělost v kladení otázek, touhu po neznámých krajích skrývajících se za obzorem jejich momentálního poznání. Jazyk Wirth se si kladl za cíl šetřit znaky potřebné k zápisu, a přitom ještě zůstat čitelným a přehledným jazykem. V dobách, kdy jsem se jím zabýval, vévodily světu programování strukturované jazyky **C++** (1985) a **Pascal** (1970). Relativní novinkou bylo objektové programování a jazyky určené pro programování webových aplikací se teprve draly na výsluní. V knihovně pardubického Střední průmyslové školy elektrotechnické a vyšší školy elektrotechnické, kde jsem tou dobou studoval, jsem objevil mezi vyřazovanými knížkami slovensky psané dílo švýcarského informatika **Niklause Wirtha** (*1934): **Algoritmy + datové struktury = programy** (1975). Tato právě odepsaná kniha, popisovala filozofii strukturovaného programování a principy, o které se opírala, deklarovala na ukázce kódu kompilátoru pro jazyk Pascal. Neumím si představit, že by podobná kniha mohla vzniknout dnes. A to z mnoha příčin. Můj návrh programovacího jazyka Wirth byla ve skutečnosti pocta vynikajícímu vědci, který stvořil na svou dobu úžasný programovací jazyk jenom proto, aby své studenty naučil programovat a nadto je naučil i rozumět tomu, co programují, jak funguje zpracování jejich příkazů, co je za tím tlačítkem kompilátor. Niklaus Wirth byl nejenom vizionář a praktik, ale především dobrý pedagog. Měly bychom si takovýchto osobností vážit a děkovat jim za jejich objetavost. Že, slečno **Augusto Ado Byronová** (1815–1852)!

Jiří (Vyšín) **Altior**, 16. května 2022, Praha
Zakladatel **Občanského Sdružení Aplikované Kreativity** (*2007)

Obsah

Úvod

Ve své seminární práci se zabývám problematikou vývoje nového programovacího jazyka. Na začátek je ale zapotřebí vysvětlit, co je programovací jazyk obecně, proč není pouze jeden univerzální, ale existuje jich více, čím se liší jednotlivé druhy jazyků od sebe a na co se používají. Rád bych hned na úvod podotknul, že problematikou vývoje umělých jazyků (mezi které se programovací jazyky bezesporu řadí) se zabývá vícero vědních oborů a realizace vývoje nového programovacího jazyka čerpá ze znalostí a poznatků mnoha, někdy i vzájemně velmi vzdálených vědních disciplín.

1 Vznik a vývoj programovacích jazyků

1.1 Historie počítačů a počítačových programů

Historie počítačových programů začíná na počátku minulého století, ale první programy jsou daleko starší. Program lze definovat jako ucelený souhrn instrukcí (příkazů), pomocí kterých je prováděna určitá činnost (v případě počítače: kterou počítač provádí určitou činnost) [VS]. Z této definice vyplývá, že za program lze počítat například i recept na vaření.

Historie počítačových programů je tedy stejně stará jako počítačů samotných, se kterými je nedílně spjata a bez kterých počítače ztrácejí smysl. První jednoduché automaty i návrh principu programového řízení automatu vznikly již v antice (Heron Alexandrijský) a v 18. století našeho letopočtu došlo k výraznému rozšíření hracích strojků s vyměnitelnými válečky, které byly vlastně prvními programy. První praktické využití programového řízení použil Francouz Jacquard kolem roku 1800: použil děrných štítků s naprogramovaným vzorem látky tkalcovského stavu. V té době mechanické výpočetní pomůcky a programově řízené stroje tvořily dvě izolované větve vývoje. Myšlenku jejich spojení navrhl a realizoval roku 1834 anglický matematik Charles Babbage (1791-1871). Navrhl programově řízený mechanický číslicový počítač, který nazval Analitical Engine. Tento počítač neměl však ve své době šanci být realizován, a po Babbageovi smrti byl zapomenut.

Roku 1890 bylo při sčítání lidu v Americe úspěšně použito děrných štítků jejich vynálezcem Hermanem Hollerithem (zakladatelem firmy IBM). Ten pomocí nich realizoval myšlenku uložení všech dat pomocí celých čísel. Během let 1937-1943 byl v laboratořích IBM vytvořen první elektromechanický počítač Mark I. a prvním čistě elektronickým univerzálně programovatelným počítačem se stal elektronkový ENIAC. Ale teprve 30. 6. 1945 byl poprvé publikován koncept počítače, který je platný dodnes. Uveřejnil jej ve svém článku First Draft of a Raport on the EDVAC John von Neumann (1903-1957) se svými kolegy a dnes je všeobecně znám jako von Neumannova koncepce počítače [CMP]. Nástupem nových technologií, použitím tranzistorů (později IO), feritových pamětí a též zavedením sériové výroby se počítače stávaly stále více přístupnějšími. A právě v té době začaly vznikat první vyšší programovací jazyky, mezi nimiž byl asi nejdůležitějším FORTRAN. Dalšími mezníky ve vývoji počítačů byl vznik mikroprocesorů (poprvé čip i4004 v roce 1971), univerzálních mikropočítačů a vzrůstající dostupnost počítačů běžným uživatelům.

1.2 Strojový kód, Assembler language

První počítače měli různý způsob programování většinou závislý na konkrétních možnostech stroje. ENIAC byl programován spojováním kontaktů v matici, později umožňovaly různé druhy pamětí ukládat program a moci jej pozměňovat bez nutnosti vytvářet novou kopii programu (přechod od děrných štítků k magnetickým paměťovým médiím). Tyto počítače byly programovány přímo ve strojovém kódu, který počítač uměl přímo vykonávat a nebyla možnost zápisu žádné abstrakce.

Při stále větším počtu programů začalo takovéto absolutní vyjadřování se ve strojovém kódu nevyhovovat, a tak se pomalu začaly používat mnemotechnické zkratky, které sice ještě stále byly úzce svázány se sadou instrukcí, ale umožňovaly zápis některých, častěji se vyskytujících instrukcí jako znakovou zkratku. Například místo instrukce 240982 zapsat ADD 4,0982. Jenže neustále se jednalo de facto o programování ve strojovém kódu, neboť programátor byl nucen přímo přiřazovat každou adresu v programu na absolutní pozici, což značně komplikovalo jak vývoj, tak i pozdější změnu programu. Použitím symbolických adres, kdy autor programu v době vývoje ještě nezná absolutní adresu a tu určí sám počítač, vznikl Assembly language (Assembler). Jedná se o nejnižší programovací jazyk, který pracuje s relativními adresami, mnemotechnickými zkratkami – zástupnými zkratkovými symboly (např. INC (inkrement), MOV (přesunutí hodnoty) nebo JMP (nepodmíněný skok)) a jehož kód je pro programátora čitelný. Název jazyka je z anglického slova assemble (sestavení, smontování) a v češtině se jazyk nazývá Jazyk symbolických instrukcí. Jak již bylo uvedeno, Assembly language zůstává stále na úrovni strojového kódu, je závislý na mikroprocesoru a není přenosný napříč různými platformami.

1.3 Vyšší programovací jazyky

1.3.1 FORTRAN

V roce 1957 vznikl u firmy IBM první použitelný vyšší programovací jazyk FORTRAN (FORmula TRANslator). Původně byl používán v oblasti vědecko technických výpočtů a byly jím reprezentovány složité matematické vztahy. Před vznikem FORTRANu byl programátor především odborníkem na použitý výpočetní systém, neboť na jeho znalosti každého detailu závisela správná funkce programu, navíc s přihlédnutím k omezeným prostředkům hardware. V polovině padesátých let došlo k značnému zvýšení výroby počítačů a dosud používaný způsob programování byl již nepoužitelný. Proto začaly vznikat vyšší programovací jazyky. Jednou ze základních myšlenek vyšších programovacích jazyků byl vznik uživatelského přístupu k počítači. Idea byla taková, že bude možné psát programy, aniž by se programátor musel zabývat problémy nesouvisejícími s řešenou problematikou. FORTRAN používá oddělenou kompilaci programových jednotek, což umožňuje vznik specializovaných jednotek řešících konkrétní, především matematické problémy, které jsou využitelné v rozličných programech. Díky velké skupině těchto jednotek se FORTRAN používá dodnes, i když v novějších verzích (FORTRAN II, IV, 66, 77, 90, FORTRAN D, FORTRAN-Plus).

Popis jazyka – způsob, jakým je definován – je neformální. FORTRAN obsahuje všechny základní typy, tzn. celočíselný Integer, reálný Real a logický Boolean. Navíc obsahuje datový typ pro komplexní číslo, což umožňuje snadnější vyjadřování matematických formulí. Nezná ale typ ukazatel (Pointer). V řídích strukturách používá přímý zápis pouze příkazu pro větvení dvou cest (IF) a všechny ostatní řídící struktury realizuje pomocí kombinace skoku (GOTO) s návěštím, příkazu DO (při realizaci cyklu) a CONTINUE. U cyklů musí být počáteční hodnota konstanta, krok cyklu jednoduchá hodnota a řídící hodnota celočíselná. Příkaz skoku zde funguje jako přepínač, což znamená, že v závislosti na proměnné předává řízení na různá místa programu. FORTRAN umožňuje počáteční inicializaci proměnných, má bohaté prostředky pro vstup a výstup (formátovací přenos, binární přenos buď rychlý nebo s mezivýsledky, sekvenční či přímý do souboru) a procedury (parametry) mohou být volány jak adresně, tak i pomocí prázdného argumentu (výpočet v lokálních datech) nebo jmény polí.

1.3.2 ALGOL

Algoritmický jazyk (ALGOrithmic Language) ALGOL byl poprvé definován v roce 1957 Peterem Naurem a J. W. Backusem. V Zürichu v listopadu 1958 a v Copenhagenu v únoru 1959 proběhly dvě významné konference, kde byl ALGOL představen odborné veřejnosti a na mezinárodní konferenci v lednu 1960 v Paříži byl vydán jeho standart pod názvem ALGOL 60. Backus s Naurem použili pro definování jazyka nový způsob zápisu formálního zápisu, který se ukázal jako velice praktický a byl později použit při definování dalších jazyků jako např. PASCAL či ADA. Tento způsob zápisu byl podle svých autorů nazván Backus-Naurova forma (zkráceně BNF) viz. kapitola 3.4.

ALGOL existuje ve více verzích, které vznikaly postupným vývojem. Nejvýznamnější z nich je první verze ALGOL 60, potom verze 62 a především 68. Ač byl ALGOL ve své době velice pokrokový a používal se zejména pro vědeckotechnické výpočty, nebyl nikdy příliš rozšířen a po verzi 68 prakticky zanikl. Zjevným důvodem byla jeho stále větší složitost, a především nepropracovaný vstup a výstup. Byl však inspiračním zdrojem pro mnoho úspěšných jazyků, které z něj vycházeli, např. v roce 1971 PASCAL, potom ADA či MODULA II. Verze ALGOL 60 a ALGOL 68 se velmi lišily.

1.3.2.1 ALGOL 60

ALGOL 60 je navržen jako počítačově nezávislý jazyk s výraznou blokovou strukturou orientovaný na vědeckotechnické výpočty. Obsahuje všechny základní datové typy a umožňuje pracovat s indexovanýma proměnnýma neboli poli, které jsou reprezentovány dynamicky. V řídících strukturách používá vedle příkazu jednoduchého větvení IF také přepínač SWITCH, který jde sice použít i v jako součást výrazu (jako IF), ale pracuje pouze jako přepínač pro návěští nepodmíněného skoku GOTO. Cyklus FOR je tvořen proměnnou a posloupností hodnot, které postupně nabývá. Mezi výčet stavů lze uvést i posloupnost ve formě Inicializační-proměnná STEP Krok UNTIL koncová-hodnota či Aritmetický-výraz WHILE podmínka. Takto zapsaný cyklus umožňuje průběh podle libovolné posloupnosti, což výrazně zpřehledňuje a usnadňuje zápis algoritmu. V ALGOLu je umožněno vícenásobné přiřazení, použití konstrukce IF THEN ELSE při vyčíslování logických i aritmetických výrazů a procedurální programování (odkazy hodnotou, jménem, jménem pole). Jak již bylo zmíněno, slabinou jazyka se staly vstupně výstupní operace, které prakticky neumožňují mimo přístupu do souboru nic.

1.3.2.2 ALGOL 68

ALGOL 68 reagoval na vývoj vyšších programovacích jazyků a snažil se vyřešit problém prostředků pro vstup a výstup. Je již univerzálně orientován jak na vědeckotechnické výpočty, tak i na zpracování hromadných dat (viz. COBOL), zpracování textů a paralelní procesy. Jeho nevýhodou se ale stalo právě příliš mnoho prostředků, takže je těžko zvládnutelný. Typy umožňují dvojnásobnou až n-násobnou přesnost, je zaveden datový typ pro komplexní číslo (podobně jako u FORTRANu) a vzniká datový typ řetězec STRING, který má proměnné meze (definován jako 1..0 tzv. flex char). Obsahuje odkaz ukazatelem (procedurou), procedurový typ a rekurzivní typy (např. prvky pole procedur). Umožňuje již také práci s vícenásobným větvením pomocí konstrukce CASE podle vyčíslené hodnoty výrazu a součástí byla i větev jinak ELSE. Cyklus používá libovolnou řídící proměnou, která může být jak skalárem, tak intervalem a vyčísluje se 1krát.

Operátory umožňují pracovat s komplexními čísly, s dolní a horní mezí pole a je možné je deklarovat. Přístup k prvkům pole jde mimo klasického způsobu výřezem (násobné hodnoty) nebo redukcí indexů (vynecháváním). Parametry jdou předávat také kopírováním. Podstatných změn doznal ALGOL 68 v prostředkách pro vstup a výstup (Logické a fyzické chyby, neomezené (volba číselné soustavy, volba alternativ atd.), binární přenos a soubory přímé či sekvenční).

1.3.3 COBOL

Americkou reakcí na evropský vývoj vyššího programovacího jazyka (ALGOL) byl v letech 1959 až 1961 vývoj rozsáhlého jazyka COBOL (Common Business Oriented Language) určeného především na zpracování hromadných dat zejména v oblasti ekonomie. Jedná se o v mnoha směrech zvláštní jazyk, který nachází své uplatnění i dnes, i když v podstatně menší míře. Jeho syntaxe vychází z anglického jazyka a z části se snaží kopírovat i jeho gramatiku. Z tohoto hlediska se jeví pro anglofonního programátora jako jazyk vcelku snadný a svým způsobem přirozený. Negativní důsledek snahy autorů (FLOW-MATIC, Programming for the Univac ® I and II, Data Automation Systems, IBM, FACT, DSI) je značná objemnost výsledného kódu, což zbytečně zpomaluje vlastní vývoj programu i jeho kompilaci [AFC]. Ve své době přinášel COBOL mnohé novinky. Systém COBOLu se skládá ze dvou částí. Z jazyka a překladače pro každý typ počítače jiného. COBOL je navržen doslova univerzálně. Idea COBOLU byla taková, že programovací jazyk je prostředek, pomocí kterého se popíše (v angličtině) operace, kterou si programátor přeje, aby byla vykonána a ta se podle toho kterého počítače převede do strojových instrukcí, přičemž jedna COBOLovská věta se často přeloží do desítek strojových instrukcí. Ještě dřív, než se může program na počítači zpracovat, se musí každý příkaz přeměnit na vnitřní jazyk počítače, a proto program napsaný v COBOLu se musí do strojového jazyka přeložit ještě dřív, než se doplní daty, které má spravovat. Tento překlad zprostředkuje COBOLovský překladač.

Jazyk COBOL je schopný popsat algoritmy hromadného zpracování dat různého druhu určením základních kroků, které jsou potřebné na jejich řešení. Postupy v něm můžou být napsány bez pochopení podrobných činností, které počítač při jejich zpracování uskutečňuje. Znalost konkrétního počítače ale umožňuje efektivnější využití jeho možností. Procedurální část jazyka je ale ve větší míře na počítači nezávislá.

COBOL nutí programátory k popisu dat a detailnímu rozboru problému, což má ve svém důsledku pozitivní vliv na kvalitu celého programu [SA]. Stavba COBOLovského programu se skládá z oddílů, které se zase člení na položky. Některé z nich můžou být volitelné, jiné jsou povinné. Vedle oddílu dat DATA DIVISION, procedur PROCEDURE DIVISION, výkonného kódu WORKING-STORAGE SECTION atd. existuje v identifikačním oddílu IDENTIFICATION DIVISION položky jako typ zdrojového počítače SOURCE COMPUTER, identifikátor programu PROGRAM-ID, autor AUTHOR, instalace INSTALLATION…

Ze standardního hlediska hodnocení jazyka je COBOL jazykem, jehož základní typy se popisují výčtem desítkových číslic, který nemá typ pointer, umožňuje definovat statické pole a hierarchický datový typ. COBOL neobsahuje přepínač CASE či SWITCH, řídící hodnota cyklu je celočíselná a neumožňuje paralelismus. Přiřazení se děje pomocí přesunu prvků se stejnými strukturami tvarem MOVE. data lze na počátku inicializovat. Procedury jsou volány adresně, nebo jménem pole a prostředky pro vstup / výstup jsou pro soubor buď standardně sekvenční či přímé nebo indexové, existuje zde výstup a ošetření chyb a REPORT WRITER, což je část překladače pro generování sestav.

1.3.4 PASCAL

V roce 1968 publikoval holandský informatik Edsger W. Dijkstra článek nazvaný O škodlivosti příkazu GOTO (GOTO Statemen Considered Harmful). V článku Dijkstra na příkladech dokazoval, jak neuvážené používání příkazu skoku v ALGOLovském programu může zatemnit smysl algoritmu a vést k nežádaným vedlejším efektům. Z Dijkstrových myšlenek vycházel profesor informatiky Niclaus Wirth, který ve svém novém jazyce, jehož popis poprvé zveřejnil v roce 1971 a nazval Pascal, vycházel z nejnovějších poznatků své doby. PASCAL (pojmenován po francouzském matematikovi Blaise Pascalovi, autorovi prvního mechanického počítacího stroje), narozdíl od jazyka FORTRAN, COBOL či ALGOL používá důslednou koncepci datových typů a strukturovaného programování. V mnohém navazuje na ALGOL, ale čerpá i z FORTRANU a COBOLU. PASCAL byl vyvinut jako jazyk pro výuku strukturovaného programování, ale rychle se rozšířil, zejména díky své přehlednosti a jednoznačnosti. Vzniklo mnoho implementací a verzí pro různé typy počítačů a operačních systémů. Významnou se stala zejména řada verzí Turbo PASCAL od společnosti BORLAND, jejíž čtvrtá verze umožňuje zejména tvorbu samostatně kompilovatelných programových jednotek (Turbo Pascal Unit, TPU) a ve verzi 5.5 jsou poprvé použity objekty [DK]. Osmou verzí Turbo či Borland PASCALu se stal Object PASCAL, použitý v Delphi. Základní novinkou je nový způsob implementace objektů, obsluha výjimek a zjišťování typu za běhu programu. Delphi již reagují na nový vývoj v oblasti operačních systémů (Microsoft Windows 95) a skládají se ze skupiny nástrojů a pomůcek určených pro vývoj programů (zde již aplikací), takže význam vlastního jazyka se mírně ztrácí. V důsledku toho vede k přibližování některých vyšších visuálních programovacích jazyků, jako jsou PASCAL v DELPHI, C ve Visual C++ nebo BASIC ve Visual BASIC.

PASCAL samotný umožňuje od počátku práce s ukazateli, statickým polem, intervalem, množinou a strukturovanými položkami. Obsahuje veškeré základní řídící struktury - jednoduché větvení IF THEN ELSE, přepínač CASE, cyklus FOR s inkrementací či dekrementací, cyklus s podmínkou na začátku WHILE i na konci UNTIL a připouští i polemický příkaz skoku GOTO. Na rozdíl od jazyka ALGOL neumožňuje použití složitějších posloupností v cyklu FOR ani vícenásobné přiřazení. Definuje operátory pro práci se všemi základními typy (včetně IS u množin) a k parametrům lze přistupovat adresně, hodnotou, jménem pole, procedurálně či funkčně. Jazyk PASCAL je definován pomocí BNF a zapsán je též syntaktickými diagramy.

1.3.5 ADA

V osmdesátých letech byl na zakázku americké armády vytvořen jazyk nazvaný po hraběnce Lovelace Augustě Adě Byronové (dcera lorda Byrona, byla první programátorkou) ADA. Tento jazyk je považován za nejdokonalejší vyšší programovací jazyk. Je vysoce strukturovaný, specializovaný na řízení procesů v reálném čase a používá se v tzv. vložených (embeddet) systémech (tj. v systémech, kde je počítač použit pro řídící účely v reálném čase) [3]. Umožňuje paralelní programování a již od začátku má některé rysy objektově orientovaného programování. Vyznačuje se efektivností cílového kódu (návaznost na ALGOL 68). Používá se na příklad ve střelách s plochou dráhou letu.

ADA umožňuje dvojnásobnou přesnost všech základních datových typů, pracuje s ukazateli (typ přístup), pole reprezentuje jako dynamickou strukturu a jako další typy obsahuje konečné množiny, výčet, intervaly, rekurzivní a procedurální typy a popisy uživatelských typů (struktury). Příkaz větvení obsahuje koncový omezovač podmíněné věty (jako ALGOL 60 / 68). Mezi řídící struktury dále patří cyklus převzatý z jazyka PASCAL a nutno sem zařadit i propracované řízení paralelních procesů. Je možné (jako později u jazyka C) deklarovat nové operátory a Též inicializovat proměnné. ADA také řeší slabé stránky blokové struktury programů (která implicitně odděluje lokální data od globálních), kdy nedostupnost objektů vnořeného bloku je pouze implicitním pravidlem. Explicitně lze přístup k těmto objektům zjednat [CMP]. Navíc ADA disponuje propracovaným systémem modulů.

1.3.6 BASIC

V roce 1964 byl vyvinut BASIC (Beginner's All-purpose symbolic instruction code – Universální symbolický operační kód pro začátečníky), který se dá velmi snadno naučit. Jazyk obsahuje několik desítek příkazů, přičemž řídící struktury jsou až na větvení a cyklus řešeny pomocí příkazy skoku GOTO anebo ON GOTO. Má nedostatečné vstupně výstupní rozhraní, neumožňuje tvorbu procedur atd. Jeho jedinou výhodou je snadnost a velká dostupnost (distribuoval se s operačním systémem MS DOS a byl velmi rozšířený na osmibitových počítačích). Kvůli nedostatečné podpoře strukturovaného programování a preferování příkazu skoku se od něj upustilo. Částečnou renesanci doznala až jeho plně strukturovaná objektová verze od Microsoftu podporující Visuální programování Visual Basic. Zajímavostí jazyka BASIC je datový typ Variant.

1.3.7 C, C++

C je programovací jazyk vytvořený Dennisem Hallem v roce 1972, který je vyvinut pro implementaci operačního systému UNIX. Standardem jazyka se stala verze autorů Brain W. Kerninghana a Denis M. Ritchieho, která byla popsána v knize The C programming language. Oficiální norma ANSI C byla vydána až roku 1988 a obsahuje mimo jiné i oficiální množinu knihovních funkcí a hlavičkových souborů, které jazyk využívá. Jazyk C je navržen jako 100% přenositelný mezi různými platformami, a přitom umožňující programovat na velice nízké úrovni. Kombinuje možnosti vyššího programovacího jazyka a assembleru. C podporuje strukturované (později i objektové) programování, členění logických celků kódu do hlavičkových souborů (obdoba knihoven u Pascalu) a knihovních funkcí (soubory *.lib). Ač jazyk C není specializován na jednu úroveň, je jednoduchý, nezávislý na počítači a má stručné vyjadřování. Za mínus jazyka se počítá fakt, že ANSI C neobsahuje nástroje pro práci s řetězci Pascalovského typu (pole znaků), což zbytečně komplikuje práci s textem. Jazyk C vyniká především v práci s ukazateli. Nadstavbou jazyka C se stal jazyk C++ (autor Bjarne Stroustrup), který byl rozšířen o objekty a kde byla zlepšena práce s funkcemi a přidána kontrola typů a tříd. ANSI C předává parametry hodnotou, C++ i odkazem. Správa paměti však není automatická a musí být prováděna programátorem. C++ je dále charakteristický virtuálními funkcemi a šablonami tříd. Díky kombinaci hutné syntaxe a velkých možností je v současnosti nejpoužívanějším jazykem, zejména při kombinaci s visuálním programováním (např. MS Visual C++ v.5).

1.4 Neprocedurální, paralelní a speciální jazyky

Výše zmíněné jazyky jsou či byly ve své době nejčastěji používanými programovacími jazyky a řadí se mezi imperativní jazyky (výjimkou je BASIC), tzn. jazyky, který důsledně rozlišují řídící struktury neboli příkazy a datové struktury, nad nimiž příkazy pracují, a v podstatě na sebe historicky navazují. Pro úplnost je ale zapotřebí se zmínit o některých dalších jazycích.

1.4.1 LISP

V době vzniku jazyka ALGOL a COBOL se objevil LISP (LISt Processing), který se řadí mezi jazyky funkcionální (jako např. PROLOG). Jedná se o plně strukturovaný jazyk, který se skládá pouze ze seznamu volání funkcí. Program v LISPu je tvořen definicemi funkcí podle pravidel a výsledkem činnosti programu je určení hodnoty nějaké funkce, aplikované na konkrétní operandy. LISP tedy nedělá rozdíl mezi programem a daty, neboť jeho příkazy se vykonávají okamžitě (interpretují se pomocí interpreteru, jako v případě jazyka BASIC). LISP nachází praktické uplatnění nejen při výzkumu umělé inteligence, ale i v praxi. Ve své variantě známé jako AutoLISP je standardním programovacím jazykem grafického editoru AutoCAD.

1.4.2 PROLOG

Do stejné kategorie jako LISP bývá řazen také PROLOG (PROgramming in LOGic) vyvinutý v sedmdesátých letech v Evropě pro logické programování. PROLOG používá jednotnou strukturu dat, zvanou term, pomocí které jsou tvořena nejen všechna data, ale i programy. Program potom tvoří množinu klauzulí, z nichž každá je buď fakt nebo pravidlo, které ukazuje, jak může řešení přímo vyplývat nebo jak může být přímo odvozeno z daných faktů. PROLOG se řadí do skupiny jazyků logického programování (logic programming). Logické programování je ve všeobecnosti metoda strukturování programů jako souborů logických pravidel s předem definovanými algoritmy, pro zpracování vstupních dat podle pravidel vstupního programu. Logický program je množina formulí a výpočet je posloupnost logických důsledků odvozená z programů a cílové klausule [4].

1.4.3 OCCAM

Pro transputerové systémy byl firmou INMOS vyvinut speciální programovací jazyk OCCAM. Je navržený na plné využití možností architektury transputerů a umožňuje v maximálně možné míře využít výkonosti transputerové sítě. Transputer je speciální jednočipový počítač, který má paměť, procesor a prostředky pro komunikaci s jinými transputery. Používá se v multiprocesorových systémech (př. systolické nebo neuronové sítě), které dosahují výkonu 10 MFLOPS na 1 transputer (výkon roste téměř lineárně s počtem transputerů) a jejichž výkonnost spočívá právě v komunikačních schopnostech transputerů. Základní idea programování v OCCAMu je: nejprve popsání typů procesorů v síti a poté jejich propojení, přičemž definice procesoru odpovídá zhruba proceduře a popis hlavnímu programu. OCCAM tedy umožňuje přímo adresovat komunikační kanály sítě a řídit tak přenos dat v síti.

1.4.4 4GL – jazyky čtvrté generace

1.4.4.1 APL, MATLAB

4GL jinak Fourth generation language (Jazyk čtvrté generace) jsou navrženy jako jazyky čtvrté generace pro málo cvičené programátory a řadí se k jazykům jako Smalltalk, APL a MATLAB (4GL je i název prvního jazyka čtvrté generace). APL (A Programming Language) je interpretační jazyk výrazný především svojí vyjadřovací silou a nepřehledností. Jeho základem je běžná kalkulačka, která ovšem v APL dokáže pracovat s vektory, maticemi apod. [1]. Podobným, novějším a lepším matematicky zaměřeným programovacím jazykem čtvrté generace je MATLAB umožňující programování s maticemi. Matice se v jazyce MATLAB vkládají po řádcích, lze je spojovat a provádět mezi či s nimi různé operace. Užitečnost MATLABu spočívá ale v jeho stavebnicovosti, takže dnes je využíván například pro složité výpočty v elektrotechnice (Signal processing toolbox) a zároveň v elegantní jednoduchosti (Příkazem X = A \ M lze vyřešit soustavu lineárních algebraických rovnic).

1.4.4.2 Smalltalk

Smalltalk byl zase první jazyk, který přinesl koncepci oken a také první, plně objektový programovací jazyk. Vyvinula jej firma Xerox a rozšířil se i na počítačích Sun. Jedná se o interpretační jazyk vhodný pro interaktivní vytváření programů a rychlé prototypování (vývoj předběžné verze softwarového systému, aby bylo umožněno prozkoumat jiné aspekty systému) [3].

1.4.4.3 SQL a databázové dotazovací jazyky

SQL (Structured Query Language) patří mezi dotazovací jazyky (Query language) pro manipulaci s daty, které se vyznačují tím, že jsou interaktivně přístupné uživatelům. Tyto jazyky jsou určeny pro vyhledávání informace a dále se dělí podle dvou základních technik na jazyky s technikou Query-by-example (SQL) a On Line English. Metoda Query-by-example předkládá uživateli na obrazovce základní rám tabulky s vypsanou strikturou věty databáze a uživatel vyplňuje podmínky, za kterých chce znát hodnotu nebo naopak, které položky mají společnou hodnotu. (Hodnota příslušného políčka, které chceme zjistit se například označí symbolem P.). Přínos jazyka spočívá v tom, že není zapotřebí vymýšlet dotaz, ale stačí předložit vzor odpovědi, která je požadována.

Jazyk typu On Line English pracuje s reálným jazykem. Nejdříve uživatel položí otázku (např. WHAT ARE THE NAMES OF PROGRAMMERS IN PARDUBICE), systém na položený dotaz zareaguje zopakováním již přetransformované otázky (v uvedeném příkladu PRINT NAME OF ALL EMPLOZEES WITH CITY = PARDUBICE AND JOB = PROGRAMMER) a je-li dotaz potvrzen, vypíše výsledek. Riziko nedorozumění je odstraněno právě zpětným dotazem. Je nutné podotknout, že vlastní zadávání dat a práce s databázemi není obsažena v těchto jazycích.

1.4.5 Ostatní programovací jazyky

Programovacích jazyků existuje v současnosti několik tisíc, přičemž neustále vznikají nové. Z toho je asi sto až dvě stě programovacích jazyků, jež jsou v některých rysech unikátní a které si zaslouží pozornost. Některé z nich jsem zde nezmínil, ač jsou progresivní a v současnosti značně rozšířené (například JAVA), jiné historicky zajímavé (snaha sjednotit FORTRAN, COBOL a ALGOL v PL/1 či FORTH, který byl vyvinutý v šedesátých letech a vyznačoval se propracovanou strukturovaností , používáním postfix notace a je dodnes používán v nekomerční sféře např. na řízení teleskopů na hvězdárnách), či byly mezistupni, bez kterých by se vývoj znatelně zpomalil (OBERON od Niclause Wirtha, Modula II, Simula, DYLAN pro počítače Apple). Velkou skupinu tvoří též implementační jazyky specializované na konkrétní druh software (většinou od jedné firmy) nebo typ problému (FoxPro, Clipper). V neposlední řadě stojí za zmínku jazyky typu Perl či Eiffel, které se používají ve Zpětném inženýrství (Japonské označení nelegálního extrahování algoritmů z cizích programů, srov. s hackerstvím).

1.5 Shrnutí vývoje a trendy programovacích jazyků

Programovací jazyky prošly bouřlivým vývojem, který se poněkud utlumil v osmdesátých letech (tzv. krize softwaru). Z počátku vládnul optimismus a věřilo se, že lze vytvořit univerzální jazyk, ve kterém se budou programovat všechny aplikace. Tato idea je ale neslučitelná s principy vývoje systémů, jejichž složitost neustále roste a tím se stupňují i požadavky (zejména šíře spektra) na univerzální programovací jazyk. Jistý posun ve vývoji jazyků v tomto smyslu však existuje, což dokazuje zejména visuální programování, kdy se vývoj aplikací na konkrétní platformu stává téměř totožným, ať se použije jakýkoliv z vyšších programovacích jazyků Visual C++, Optima, Visual BASIC či Delphi. Tyto jazyky se liší zejména použitou syntaxí (gramatikou). Každý z nich obsahuje své silné stránky, ve kterých mírně dominuje (Delphi práce s řetězci a přehlednost práce s objekty, C++ silné výrazové prostředky a Visual BASIC jednoduchost). Budoucnost v tomto směru naznačuje vývoj takových jazykových překladačů a interpreterů, které budou zvládat více druhů syntaxe a budou spojovat podle aktuální potřeby výhody toho kterého konkrétního jazyka a jeho realizačního programu.

U jazyků neprocedurálních, expertních (zaměřených například na umělou inteligenci) a síťových je dána jejich odlišnost buď účelem, k jakému mají sloužit, nebo principy, které se snaží implementovat. Zde se bude dále teorie programovacích jazyků a teorie matematického programování dále slučovat s potřebami reálně existujících systémů, což vede k neustálému vývoji, který může doznat zásadní změny pouze fyzikální změnou principu strojů (např. von Neumannova koncepce), na které jsou implementovány (kvantové procesory…).

Samostatnou kapitolou jsou jazyky, které neaspirují na univerzálnost, ale na efektivnost zápisu či vyjadřovacích prostředků a které jsou určeny na řešení konkrétních (tzn. Ne universálních) problémů (např. Makrojazyky).

2 Návrh jazyka Wirth

2.1 Životní cyklus software

Jako každý rozsáhlejší programový produkt i programovací jazyk prochází během své existence několika typickými vývojovými fázemi, jejichž souhrn se označuje jako životní cyklus software a člení se na tři základní etapy: vývoj, údržbu a vyřazení. Vývoj se dělí na pět fází: 1. Stanovení cílů, 2. specifikace vnějších požadavků, 3. Návrh systému, 4. Realizace a 5. Testování a předání; údržba na 1. Opravu chyb, 2. přizpůsobení a 3. Zdokonalování. Vyřazení software se týká zejména těch produktů, které jsou principielně závislé na vnějších podmínkách (např. automatizační procesy navržené pro konkrétní časově omezené systémy).

Etapa Stanovení cílů představuje rozhraní mezi organizační a programátorskou činností. Zde se jedná o formální zadání úlohy a vyjasnění základních představ o funkcích budoucího systému, o tom, jaká omezení je třeba respektovat, jaký bude okruh uživatelů, počítačové i nepočítačové vazby na jiné systémy a také řešení praktických otázek typu termín realizace a předání, náklady apod. Další fází vývoje je specifikace vnějších požadavků. Tato fáze má za výsledek podrobný popis budoucího systému v termínech jeho vstupů a výstupů. jedná se velmi náročnou fázi, neboť se musí popsat systém, který ještě neexistuje, a přitom popis nesmí ztratit logickou celistvost. Specifikace obsahuje především jasné vymezení, co se v budoucím systému z hlediska uživatele smí a co nesmí, čili co je pro uživatele viditelné a přístupné. Forma specifikace je například v případě vývoje kompilátoru referenční manuál se slovním či grafickým upřesněním a vyjádřením.

Etapa návrhu systému plní funkci rozhraní mezi popisem systému a jeho realizací. Ujasňují se během ní metody dosažení požadovaného výsledku neviditelné pro koncového uživatele a vymezují všechny funkce systému, které se nenacházejí ve vnější specifikaci. Realizuje se dekompozicí systému na subsystémy a záleží zde na každém kroku, neboť chyby zde vzniklé se většinou projeví až po vytvoření celé aplikace při ladění anebo v horším případě až po jejím odevzdání. Dále se v návrhu subsystému stanoví způsob komunikace subsystémů mezi sebou a rozhodne organizace hlavních – především sdílených – datových struktur a rozdělí veškerá přístupová práva (tzn. které subsystémy smějí manipulovat s kterými daty). Součástí návrhu je také případně výběr metod a forem testování, jakož i stanovení pravidel pro programování (dodržované konvence).

Realizace obsahuje vypracování detailních algoritmů, sestavení programů a ladění nižších celků. Testování je poslední etapou vývoje programu. Člení se obvykle na testování subsystémů a testování celku. U testování subsystémů lze použít buď metodu shora dolů nebo zdola nahoru. Údržba systému se stará o odstranění skrytých chyb, vývoj zlepšujících funkcí nebo celých nových verzí (které mají vlastní životní cyklus) a přizpůsobení novým podmínkám.

2.2 Stanovení cílů

Během odborné praxe jsem řešil problém vývoje nového programovacího jazyka, který jsem pracovně nazval Wirth po autorovy velice úspěšného jazyka PASCAL. Jazyk je navrhnut pro zpracování logických a matematických úloh, jež realizuje speciální interpret. Interpret je součástí aplikace, která generuje přerušení. Tyto vstupní přerušení zpracovává interpret na základě připojeného programu (tzv. obslužný kód přeložený do posloupnosti instrukcí). Interpret v případě potřeby generuje výstupní přerušení, na která reaguje aplikace. Po provedení reakce aplikace vrací řízení interpretu tak dlouho, dokud není vykonán celý obslužný kód. Jazyk Wirth je vytvořen na programování tohoto obslužného kódu.

Program napsaný ve Wirthu se skládá z posloupnosti bloků, které jsou identifikovány jako vstupních přerušení a vlastních funkcí, které je možné volat pouze v rámci interpretu. To znamená, že neexistují funkce ani vstupní přerušení, který by byly volány jako první po spuštění programu, ale naopak program se skládá z množiny vstupních přerušení, která reagují na vnější požadavky.

Základním kritériem jazyka Wirth je, aby se jednalo o vyšší programovací jazyk umožňující pracovat se všemi základními řídícími strukturami (cyklus, logické větvení, příkaz nepodmíněného skoku), umožňoval pracovat s dynamickými daty, obsahoval všechny základní datové typy (celé číslo, reálné číslo, znak, logická hodnota, řetězec) a umožňoval pracovat s poli a strukturami. Jazyk naopak nesmí obsahovat žádné prostředky pro vstup a výstup dat, mimo přístupu do tabulky sdílených proměnných. Jazyk musí být maximálně stručný (ale ne na úkor přehlednosti), nezávislý na žádném přirozeném jazyce (tzn. nesmí obsahovat slova přirozeného jazyka např. angličtiny) a pokud je to možné, měl by být navržen tak, aby přechod na něj z jiného vyššího programovacího jazyka byl co nejsnazší. Jazyk Wirth by měl být principielně hardwarově nezávislý a je žádoucí, aby umožňoval rychlou kompilaci.

2.3 Specifikace vnějších požadavků

Jelikož podrobný popis jazyka, specifikace jeho vstupů a výstupů ve smyslu softwarového inženýrství (viz. kapitola 3.1) a [13] není možné v rámci rozsahu seminární práce uvést celý, zmíním se zde o hlavních rysech, způsobu návrhu jazyka a formě jeho zápisu a vlastním jazyce. Prvky, ve kterých se jazyk Wirth neliší od ostatních vyšších programovacích jazyků nejsou rozebírány a lze je vyčíst z BNF uvedené v příloze k seminární práci.

Analýzou stávajících počítačových jazyků jsem dospěl k přesvědčení, že žádný z nich nevyhovuje výše stanoveným podmínkám a že bude zapotřebí vytvořit nový. Pro definici jazyka jsem použil formát BNF, který byl poprvé použit při definici jazyka ALGOL 60 (viz. kapitola 2.3.1) a později u jazyků PASCAL, PL/0…

2.4 Backusova-Naurova forma

Počítačové jazyky jako ALGOL jsou jazyky v pravém slova smyslu. Obsahují znaky, slova, interpunkční znaménka atd. Syntaxe jazyka se popisuje metajazykem, který se skládá z několika vlastních metasymbolů a pravidel a stojí nad definovaným jazykem [14]. Jazyk (v mém případě Wirth) se zapisuje pomocí objektů a formulí. Tyto metajazykové formule obsahují metajazykové konstanty, proměnné a metajazykové spojky.

Metajazykové konstanty jsou základní symboly BNF metajazyka. Ostatní objekty, jako jsou výrazy, proměnné, operátory, komentáře atd. jsou metajazykové proměnné a zapisují se do tzv. metajazykových závorek < a >. Metajazykové spojky jsou ::= ve významu "je definovaný" a | ve významu "nebo". Spojkou ::= se spojují levá a pravá strany formule. Nalevo od něj se zapisuje definovaný objekt a napravo objekty, pomocí kterých je levostranný objekt definovaný. BNF se vyznačuje používáním rekurzivních definicí, takže je možné na pravé straně použít objekt definovaný na levé straně formule.

Oproti standardní definici jsem jazyk Wirth zapsal pomocí poněkud upravené verze BNF, použité například při definování Turbo PASCALu verze 5.0 [9]. Rozdíl spočívá jak v metaznaku ::=, který je nahrazen stručnějším =, tak i ve způsobu uzavírání objektů do metajazykových závorek. Použil jsem jako identifikátor objektu slovo neobsahující mezeru, která se v případě potřeby nahradí znakem podtržítko (případně pomlčka). Základnímu symbolu (v mém případě element jazyka) naopak bezprostředně předchází znak #, takže je zjevně patrné, který identifikátor je objekt a který základní symbol. Tím je docíleno stejné, (podle mě dokonce větší) přehlednosti a zachována původní myšlenka BNF. Dále jsem se vyhnul rekurzivnímu způsobu zápisu, který je špatně čitelný a použil způsob zápisu definice členěním na menší logické celky, přičemž jsem použil nových metajazykových závorek { a } ve smyslu "nemusí, ale může následovat buď jednou a nebo opakovaně" vztahujícím se k objektům uzavřeným v závorkách. Z důvodů přehlednosti jsem dodržel pravidlo, že budou metaznakové závorky použity v jedné formuli maximálně jedenkrát.

2.5 Syntaxe jazyka Wirth

Jako výchozí jazyk pro jazyk Wirth jsem použil ALGOL 60, PASCAL a jazyk C. Cenné podměty jsem čerpal také z jazyků ADA a BASIC. Některé prvky jazyka jsou ale naprosto odlišné (funkce vracející více parametrů, přístup k proměnným, realizace větvení typu přepínač) a jiné jsou speciální z důvodu stanovených podmínek (vstupní a výstupní přerušení, operátor Tabulka proměnných a typů apod.) Jazyk Wirth se skládá ze základních symbolů zvaných elementy jazyka (dále jen elementy) a identifikátorů, které se v jazyce Wirth se můžou skládat z písmen velké a malé abecedy, číslic a znaku podtržítka, přičemž číslice nesmí být na prvním místě. Elementy jazyka jsem z důvodu stručnosti jazyka a rychlosti kompilace určil maximálně dlouhé dva znaky

.

2.5.1 Identifikátory

Wirth rozlišuje tyto identifikátory: Identifikátor proměnné, funkce, vstupního přerušení, výstupního přerušení, rezervovaného slova a návěští. Vyskytuje-li se identifikátor v programu mimo definici, znamená to, že se jedná o proměnnou. Identifikátoru funkce vždy předchází znak @, identifikátoru vstupního přerušení znak $, identifikátoru výstupního přerušení dvojznak $$, identifikátoru návěští % a rezervované slovo je uvozeno znakem apostrof. Mezi elementy jazyka nelze dále počítat řetězec, který je uzavřen do uvozovek.

2.5.2 Struktura a forma jazyka

Jazyk Wirth je posloupností elementů oddělených znakem mezera nebo tabelátorem a kompilátor. Jelikož jsem musel při návrhu jazyka uvažovat i nad způsobem kompilace, abych dosáhnul pokud možno nejrychlejšího překladu, rozčlenil jsem vlastní funkci syntaktického kompilátoru na dvě části: identifikátor elementů jazyka (který je převede do ještě stručnějšího jednoznakového symbolu elementu) a překladač Backusovy-Naurovy formy, který pracuje již s takto upravenými elementy (včetně rozpoznání a přiřazení jednoznačného čísla identifikátorům a oddělení řetězců a vynechání identifikátorů). Tímto způsobem docílím velice rychlé kompilace, navíc s přihlédnutím k tomu, že druh identifikátoru je určen znakem, který mu předchází, a tak je možné vnitřně oddělit identifikátory podle typu do samostatných dynamických struktur a tím urychlit jejich vyhledávání v průběhu kompilace. Dalším výrazným rysem jazyka je dodržení požadavku, že jazyk nemá být závislý (odvozený) na žádném přirozeném jazyce. Toho je docíleno vhodným zápisem elementů jazyka, které až na dvě výjimky neobsahují znaky abecedy. Výjimkou je znak E reprezentující symbol exponent v reálném čísle a znaky A, B, C, D, E, F při reprezentaci šestnáctkového čísla. Tyto znaky patří mezi univerzální matematické vyjadřovací symboly a nejsou vázané na žádný konkrétní přirozený jazyk, takže je splněna výše zmíněná podmínka Navíc hexadecimální číslo je též uvozeno (znakem `), takže nemůže dojít k situaci, že by po načtení prvního znaku nevěděl kompilátor, s čím pracuje.

Jak jsem již nastínil, při specifikaci vnějších požadavků jsem musel použít několik prvků etapy návrhů systému. Stanovil jsem druh a typ překladače jazyka Wirth, abych docílil efektivity překladu zdrojového kódu. Musel jsem ale zvážit, zda ten který krok neomezuje možnosti budoucího programátora a zda není na úkor přehlednosti, což je jeden z požadavků na jazyk. Domnívám se, že uvození všech identifikátoru specifickým znakem umožňuje v již tak stručném jazyce docílit přehlednost. Na první pohled je totiž zřetelné, zda identifikátor označuje funkci či proměnnou nebo přerušení či klíčové slovo. Odbourává se tím potřeba vytvářet tzv. štábní strukturu při tvorbě identifikátorů, která říkala, že např. proměnné budou psány malými písmeny, funkce velkými atd. Je samozřejmé, že i tak je vhodné pojmenovávat identifikátory vystižně, stručně, víceslovné označení rozdělit podtržítkem apod.

2.5.3 Příkazy

Jazyk Wirth obsahuje tyto příkazy (viz. příloha BNF jazyka Wirth):

1) Prázdný příkaz ;
2) Příkaz jednoduchého větvení ? (tzv. IF), který po vyhodnocení podmínky jako pravdivá provede příkaz1 (tzv. větev THEN, tak) nebo příkaz2 je-li uvedený za elementem // (tzv. ELSE, jinak).
3) Příkaz složitějšího větvení ?? (tzv. CASE OF v PASCALu, nebo SWITCH v jazyku C). Tento příkaz se liší od standardního pojetí příkazu jako přepínače mezi konkrétními hodnotami, které může nabýt řídící proměnná, tím, že porovnává řídící proměnnou s pravou částí výrazu pomocí libovolného relačního operátoru. Pokud je výraz v příkazu CASE vyhodnocen jako pravdivý, následující libovolný (tedy příkaz bloku) se vykoná a další výrazy v příkazu se již nevyhodnocují a program pokračuje za vyhodnocením dalšího výrazu. Jestliže je žádoucí, aby se další příkaz (závislý na vyhodnocení výrazu) prováděl bez vyhodnocení, použije se příkazu *> (Continue, pokračuj) a jestliže je žádoucí dále již výrazy nevyhodnocovat <* (Break, přerušení) dále vyhodnocovali, použije se v příkazové části příkaz *> (zvaný Continue, přeruš), který provádění příkazu CASE ukončí.
4) Příkaz cyklu s podmínkou na začátku ?* (tzv. WHILE) a příkaz cyklu s podmínkou na konci *? (tzv. UNTIL, dokud) a provádí se tak dlouho, dokud není vyhodnocena podmínka jako pravdivá. Příkaz cyklu s podmínkou na konci provede všechny příkazy, které mu předcházejí až po element {* (tzv. Repeat, opakuj). Definice jazyka umožňuje použít též příkaz začátek bloku { .
5) Příkaz nepodmíněného skoku => (tzv. GO TO, jdi na), za kterým následuje identifikátor návěští.
6) Příkaz začátek bloku { (tzv. Begin) a konec bloku } (tzv. End). Tyto příkazy společně tvoří složený příkaz. což je posloupnost příkazů uvozená příkazem začátku bliku a ukončená příkazem konec bloku.
7) Příkaz přiřazení, který se skládá z levé strany výrazu, kde se nachází proměnná (nebo identifikátor funkce bez uvozovacího znaku @), element = a pravé strany, která obsahuje jednoduchý výraz nebo funkci stejného typu, jako je proměnná před elementem rovná se.
8) Příkaz dekrementace -- a inkrementace ++ (ve formě proměnná-- nebo proměnná++).
9) Příkaz pokračuj *> a přeruš <*, které umožňují ukončit nebo znovu vyhodnotit výraz v cyklech a příkazu složitějšího větvení.
10) Příkaz typového určení proměnné >> a příkaz strukturální proměnné << .
11) Příkaz cyklu !! (viz. For, cyklus s předem známým počtem průchodů). Strukturu tohoto příkazu přejal jazyk Wirth z jazyka ALGOL 60. Počet, kolikrát se příkaz v cyklu provede je určen posloupností hodnot, které postupně nabývá řídící proměnná. Tyto hodnoty mohou být zapsány jak výčtem hodnot (vzájemně odděleny čárkou, můžou býti použity i aritmetické výrazy), tak posloupností kroků určených počáteční hodnotou, krokem a hodnotou či podmínkou, dokdy se má krok provádět, a nebo podmínkou, dokdy se má cyklus provádět. Všechny tři možné zápisy posloupnosti nabývajících hodnot lze libovolně kombinovat, a tak vytvořit a zapsat libovolně složitou posloupnost stručně a jasně a přehledně. Příkaz kroku je !, krok smí být aritmetický operátor a jednoduchý aritmetický výraz (např. +36 nebo - x + y ** (x % y)) a podmínka je uvozena elementem !? .

2.5.4 Výrazy jazyka

Výrazy jsou totožné se všemi základními výrazy vyšších programovacích jazyků (plus, mínus, krát, děleno). Celočíselné dělení (tzv. Div) je považováno za normální dělení (aritmetický operand /), přičemž je-li první i druhý argument celé číslo, výsledek bude též celé číslo bez zbytku. Zbytek po dělení (tzv. Mod, modulus) se realizuje pomocí operátoru %.

Wirth podobně jako ALGOL umí pracovat s mocninami libovolného řádu, a to podle následujících pravidel. Jestliže označíme i jako celé číslo, r jako číslo reálné a a jako libovolné číslo potom umocnění se uskutečňuje podle těchto pravidel: pro a**i, když i>0 a a je libovolné: a*a*..*a i-krát, typ jako a když i=0, a se nerovná 0: 1, typ jako a, když i je menší nebo rovno nule a a = 0: nedefinované a když i je menší jak 0 a a se nerovná 0: 1/a*a*..*a i-krát, reálný typ výsledku; pro a**r, když a je větší jak nula: výsledek je exp(r*ln(a)) reálného typu, když a rovná s nule a r je větší jak nula, tak výsledná hodnota je nula reálného typu a jinak je výsledek nedefinován. To znamená, že například výraz 3**2 je roven 9, 3.0 ** 2 je roven 9.0, 7 ** 2.5 je roven výrazu exp(2.5 ** ln (7)) a výraz (-3) ** 2.0 není definovaný.

Relační operátory jsou shodné s relačními operátory jazyka PASCAL. Při vyhodnocování výrazů je možné použít seznamu proměnných např. min <= x,y,z <= max a používat zřetězené výrazy. Vyhodnocení se děje zleva doprava a jeli použit seznam proměnných, výraz se provede jako několik výrazů vzájemně spojených logickým AND.

Logické operátory v jazyce Wirth jsou & ve smyslu AND, | jako logický OR, |& jako XOR a negace ~. Logické hodnoty jsou :> pravda (TRUE) a :< nepravda (FALSE).

2.5.5 Datové typy, řetězce a komentáře

Datové typy se v zapisují pomoci rezervovaných slov a příkazu >>. Jazyk Wirth obsahuje všechny základní datové typy, tj. celočíselný 'Int, reálný 'Real, logický 'Bool a znakový 'Char. Navíc umožňuje pracovat indexovanými proměnnými ve formě polí a srukturovanými proměnnými, které se deklarují pomocí příkazu <<. Seznam obsažených proměnných je brán jako blok (uzavřen ve složených závorkách). To umožňuje definovat proměnné, které jsou v rámci bloku na sebe vzájemně vázané (jako např. větvení v závislosti na hodnotě atd.)

Řetězec je definován (podobně jako v PASCALu) jako pole znaků (znak se uzavírá do uvozovek) a znak uvozovky se zapisuje zdvojeně. Znak vyjádřený svojí hodnotou v ASCII tabulce se zapisuje #číslo_znaku.

Komentáře v jazyce Wirth jsou buď blokové, ohraničené (* ve významu začátek bloku a *) jako konec bloku, přičemž blok komentářů je ukončen prvním výskytem *), nebo do konce řádku uvozené elementem \\ .

2.6 Návrh systému, realizace a testování

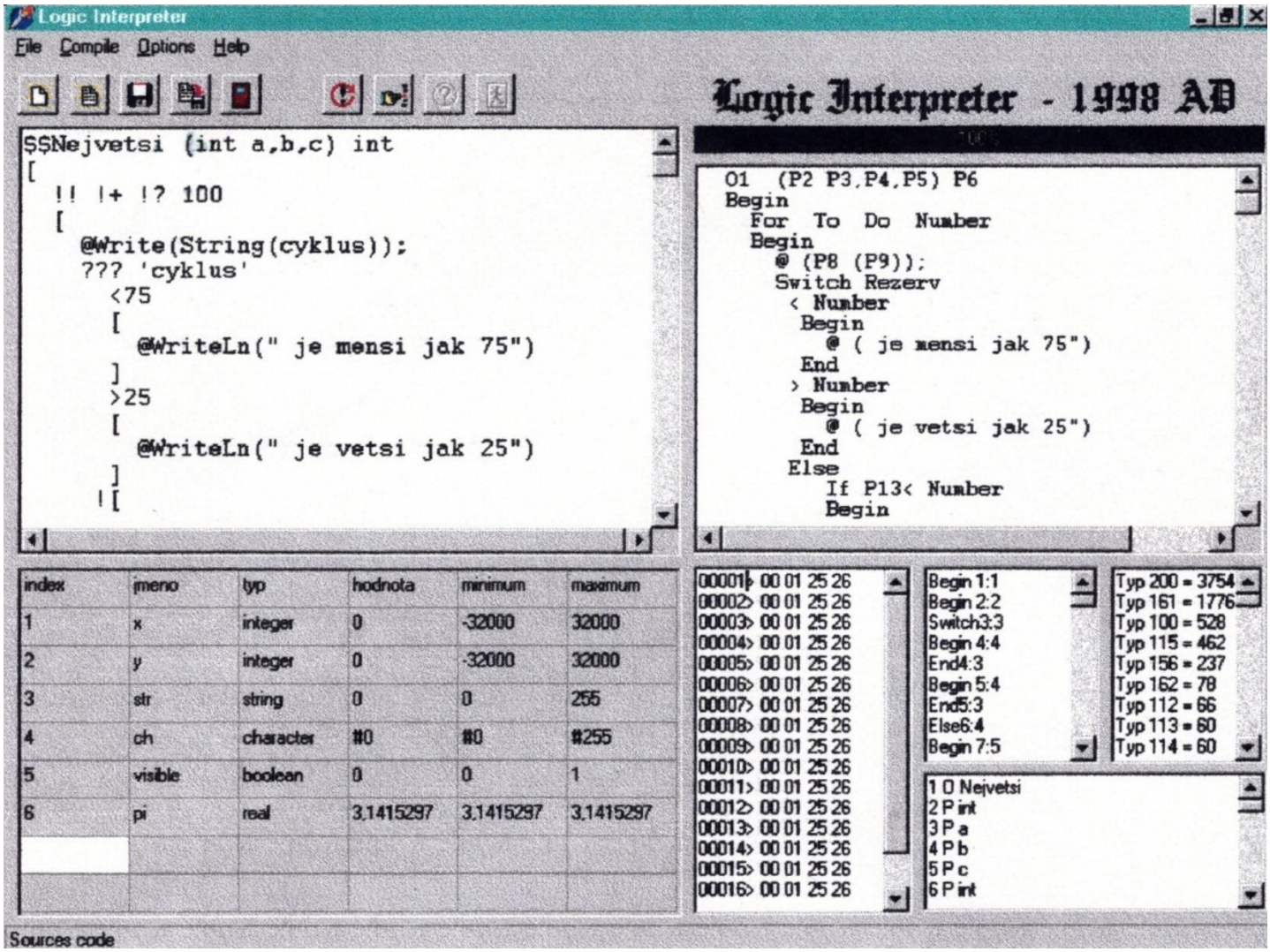
Vlastní návrh systému se skládá z podrobného stanovení vnitřních mechanismů jazyka, stanovení konkrétního typu kompilátoru (jednoprůchodový, shora dolů, s načtením jediného znaků atd. viz [6]). Realizace se zabývá vytvořením vlastního kompilátoru a testování jeho odladěním. Návrh systému, realizace ani testování není náplní seminární práce.

3 Závěr

Jak je patrné z 2. kapitoly, existuje několik základních druhů a typů počítačových programovacích jazyků. Jazyky lze členit na procedurální a neprocedurální, logické, implementační, expertní, makrojazyky… Při množství již existujících jazyků se naskýtá základní otázka, jestli má smysl vyvíjet nový programovací jazyk a zdali místo toho nestačí použít či modifikovat již stávající? Odpověď je závislá na funkci, kterou má jazyk realizovat. Jisté však je, že vzhledem k náročnosti vývoje nového jazyka se vyplatí, pokud je to možné, použít již existující.

Ve druhé části seminární práce se zabývám ukázkou postupu vývoje vyššího programovacího jazyka Wirth. Z ní je zřejmé několik věcí. Nejdůležitějším faktem je, že při řešení složitějších problémů je zapotřebí jasně formulovat co má býti vyvíjeno, čeho se má docílit a jak to má fungovat. V případě programovacího jazyka je vhodné vytvořit referenční manuál, kde bude vlastní jazyk definován (např. pomocí BNF) a jasně vysvětlit všechny sporné momenty a nejasnosti. Důležitá je také analýza a znalost již stávajících podobných produktů. Vlastní tvorba, v mém případě programování překladače má již zpracovávat takto připravenou problematiku, navíc nebyla-li vynechána etapa návrhu systému, je programování již pouhou realizací existujícího (byť na papíře). Tím se sníží riziko změny systému v závislosti na fyzických a možnostech.

Jazyk Wirth byl pečlivě navrhován půl roku. Vychází ze zkušeností a praktického programování v jiných programovacích jazycích a jejich silné stránky v mezích možností přejímá a spojuje. Je přirozeně obrazem svého zadání a tím se liší. Jeho největší výhodou je stručnost zápisu a zároveň přehlednost (jakékoliv rozhodování je uvozeno elementem s otazníkem, for vykřičníkem atd.). Datové typy jsou fyzicky zapsány v dynamické tabulce, takže je lze libovolně měnit, zjišťovat jejich hodnotu i atributy (maximum, minimum, přednastavenou hodnotu, velikost v paměti atd.). Výhodou je také skrovná slovní zásoba jazyka (57 elementů a klíčová slova pro práci s typy).



*Obrázek 1: Uživatelské rozhraní **Logického interpreteru** – překladače do/z jazyka Wirth. Na snímku obrazovky z 18. srpna 1998 je zachycen moment překladu kódu z jazyka Wirth do pseudokódu popisujících instrukce překladače, hodnot registrů, proměnných atd*

4 Seznam použité literatury

[1] **Koubský, P.**: Cesty moderního programování, 2. vydání Praha, Grada 1991, 320 s.

[2] **Havelka, J. a kol.**: Výkladový slovník výpočetní techniky a komunikací, 3. vydání Praha, Computer Press 1997, 452 s.

[3] **Vitovský, Ant.**: Anglicko-český výkladový slovník software, 3. vydání Praha, AU Software 1994, 364 s.

[4] **Vavřín, P. a kol.**: Malá enciklopedie elektroniky-Automatizační technika, 1. vydání Praha, SNTL – Nakladatelství technické literatury 1983, 664 s.

[5] **Herout, P.**: Učebnice jazyka C, 3. vydání České Budějovice, KOPP 1994, 269 s.

[6] **Wirth, N.**: Algoritmy a štruktúry údajov, 1. vydání Bratislava, Alfa 1988, 481 s.

[7] **Mikula, P.**: PASCAL 7.0 od příkladů k příkazům, 1. vydání Praha, Grada 1993, 528 s.

[8] **Sedláček, J. – Slaba, J.**: DELPHI v kostce, 1. vydání Praha, BEN 1997, 448 s.

[9] **Řepa, V. Ing. – Stanovská, I. Ing.**: Turbo Pascal v. 5.0, 1. vydání Praha, Vysoká škola elektrotechnická v Praze 1991, 180 s.

[10] **spol. Borland International**: Turbo Pascal Owners Handbook, 1. vydání Scotts Valley, USA, Borland International 1987, 654 s.

[11] **spol. Borland International**: Turbo Prolog Owners Handbook, 1. vydání Scotts Valley, USA, Borland International 1985, 218 s.

[12] **Jandoš, J.**: Programování v jazyce GW Basic, 1. vydání Praha, NOTO Kancelářské stroje 1988, 161 s.

[13] **Tieze, P.**: Strukturální analýza, 1. vydání Praha, Grada 1992, 228 s.

[14] **Jankovič, V.**: ALGOL-FORTRAN-COBOL, 2. vydání Bratislava, Alfa 1976, 373 s.

5 Přílohy

Příloha č.1: Backusova-Naurova forma jazyka Wirth – definice

```
Aditivni_Operator       = '#+ | #- | #| | #x';

Cast_Else               = 'Nic | #e Prikaz';

Cislice                 = '#1';

Cele_Bez_Znamenka       = 'Posloupnost_Cislic | #` Posloupnost_Hexa_Cisl';

Cislo_Bez_Znamenka      = 'Cele_Bez_Znamenka | Realne_Bez_Znamenka'; {!!!}
                                                {OSETRIT POUZE PRO E}

Element_Case            = 'Relacni_Operator Slozeny_Vyraz Prikaz';

Element_Seznamu_For     = 'Slozeny_Vyraz | Step_Until | For_Until';

Exponent                = 'Posloupnost_Cislic | Znamenko
                          Posloupnost_Cislic';

Faktor_Se_Znamenkem     = 'Faktor | Znamenko Faktor';

Faktor                  = 'Promnena | Konst_Bez_Znamenka | #( | Vyraz | '+
                          '#) | Konstruktor_Funkce'; {vynechana mnozina}

For_Until               = 'Jednoduchy_Vyraz #q Vyraz';

Funkce                  = 'Hlavicka_Funkce Slozeny_Prikaz';

Hexa_Cislice            = '#1 | #_'; {!!!}   {OSETRIT POUZE PRO ABCDEF}

Hlavicka_Funkce         = 'Identifikator_Funkce | Identifikator_Funkce #t '+
                          'Typ | Identifikator_Funkce #(Skupina_Parametru'+
                          ' { #, Skupina_Parametru } #) #t Typ';

Identifikator           = '#_ { Pismeno_nebo_Cislice }';

Identifikator_Fce       = 'Identifikator'; {Pouhy identifikator}

Identifikator_Funkce    = '#@ Identifikator';

Identifikator_Input     = '#$ Identifikator';

Identifikator_Navesti   = '#: Identifikator';

Identifikator_Output    = '#o Identifikator';

Identifikator_Promnen   = 'Identifikator';

Identifikator_Konst     = 'Identifikator';

Identifikator_Slozky    = 'Identifikator';

Indexovana_Promnena     = 'Promnena_Pole #[ Vyraz { #, Vyraz } #]';

Jednoduchy_Prikaz       = 'Prikaz_Prirazeni | Prikaz_Podprogram |
```

```
                    Prikaz__Goto'+' | Prazdny_Prikaz';

Jednoduchy_Vyraz    = 'Term { Aditivni_Operator Term }';

Komponenta_Promnene = 'Indexovana_Promnena | Konstruktor_Slozky';

Konst_Bez_Znamenka  = 'Cislo_Bez_Znamenka | Retezec |
                      Identifikator_Konst';

Konstanta           = 'Cislo_Bez_Znamenka | Znamenko Cislo_Bez_Znamenka
                      | '+'Identifikator_Konst | Znamenko
                      Identifikator_Konst | '+ 'Retezec';

Konstruktor_Funkce  = 'Identifikator_Funkce | '+
                     'Identifikator_Funkce #( Skutecny_Parametr '+
                     '{ #; Skutecny_Parametr } #)';

Konstruktor_Slozky  = 'Promnena_zaznam #. Identificator_Slozky';

Krok                = 'Jednoduchy_Vyraz | Znamenko Jednoduchy_Vyraz';

Multipl_Operator    = '#* | #/ | #% | #&';

Navesti             = 'Identificator';

Nic                 = '';

Pismeno_nebo_Cislice = '#_ | #1';

Posloupnost_Cislic  = '#1 { #1 }';

Posloupnost_Hexa_Cisl = '#Hexa_Cislice { #Hexa_Cislice }';

Podmineny_Prikaz    = 'Prikaz_If | Prikaz_Case';

Podprogram          = 'Preruseni_Input,Preruseni_Output,Funkce';

Prazdny_Prikaz      = 'Nic';

Preruseni_Input     = 'Identifikator_Input Slozeny_Prikaz';

Preruseni_Output    = 'Identifikator_Output Slozeny_Prikaz';

Prikaz              = 'Jednoduchy_Prikaz | Strukturovany_Prikaz';
```

Prikaz_Case = '#c Slozeny_Vyraz Element_Case { Element_Case } Cast_Else #}';

Prikaz_Cyklu = 'Prikaz_While | Prikaz_Repeat | Prikaz_For';

Prikaz_For = '#f Ridici_Promnena #= #Seznam_For Prikaz';

Prikaz_Goto = '#g Navesti';

Prikaz_If = '#? Vyraz Prikaz Cast_Else';

Prikaz_Podprogram = 'Identifikator_Input | Identifikator_Output | '+ 'Identifikator_Input #(skutecny_parametr { #, '+ 'Skutecny_Parametr } #) | Identifikator_Output '+ '#(skutecny_parametr { #, Skutecny_Parametr } #)';

Prikaz_Prirazeni = 'Promnena #= Vyraz | Identificator_Fce #= Vyraz';

Prikaz_Repeat = '#r Prikaz { #; Prikaz } #u Vyraz';

Prikaz_While = '#w Vyraz Prikaz';

Promnena = 'Uplna_Promnena | Komponenta_Promnene';

Promnena_Pole = 'Promnena';

Promnena_Zaznam = 'Promnena';

Realne_Cislo = 'Posloupnost_Cislic #. Posloupnost_Cislic | '+ 'Posloupnost_Cislic #_ Posloupnost_Cislic | '+ 'Posloupnost_Cislic #. Posloupnost_Cislic #_ Posloupnost_Cislic';

Relacni_Operator = '#< | #> | #= | #n | #m | #v';

Relacni_Otazka = 'Jednoduchy_Vyraz | Relacni_Operator Jednoduchy_Vyraz';

Retezec = '#" Znak #"';

Rezervovane_Slovo = '#'' Identifikator';

Ridici_Promnena = 'Identifikator_promnen';

Seznam_For = 'Element_Seznamu_For { #, Element_Seznamu_For }';

Seznam_Promnenych = 'Promnena { #, Promnena }';

Seznam_Identifikatoru = 'Identifikator { #, Identifikator }';

Seznam_Typu = '#{ Skupina_Parametru { #; Skupina_Parametru } #}';

Skutecny_Parametr = 'Vyraz | Promnena';

Skupina_Parametru = 'Seznam_Identifikatoru #t Rezervovane_Slovo';

Slozeny_Faktor = 'Faktor_Se_Znamenkem | #~ Faktor_Se_Znamenkem';

Slozeny_Prikaz = '#{ Prikaz { #, Prikaz } #}';

Slozeny_Vyraz = 'Jednoduchy_Vyraz { #, Jednoduchy_Vyraz }';

Step_Until = 'Jednoduchy_Vyraz #! Krok #q Relacni_Otazka';

Strukturovany_Prikaz = 'Slozeny_Prikaz | Podmineny_Prikaz | Prikaz_Cyklu';

Term = 'Slozeny_Faktor { Multipl_Operator Slozeny_Faktor }';

Typ = 'Rezervovane_Slovo | Typ_Struktura';

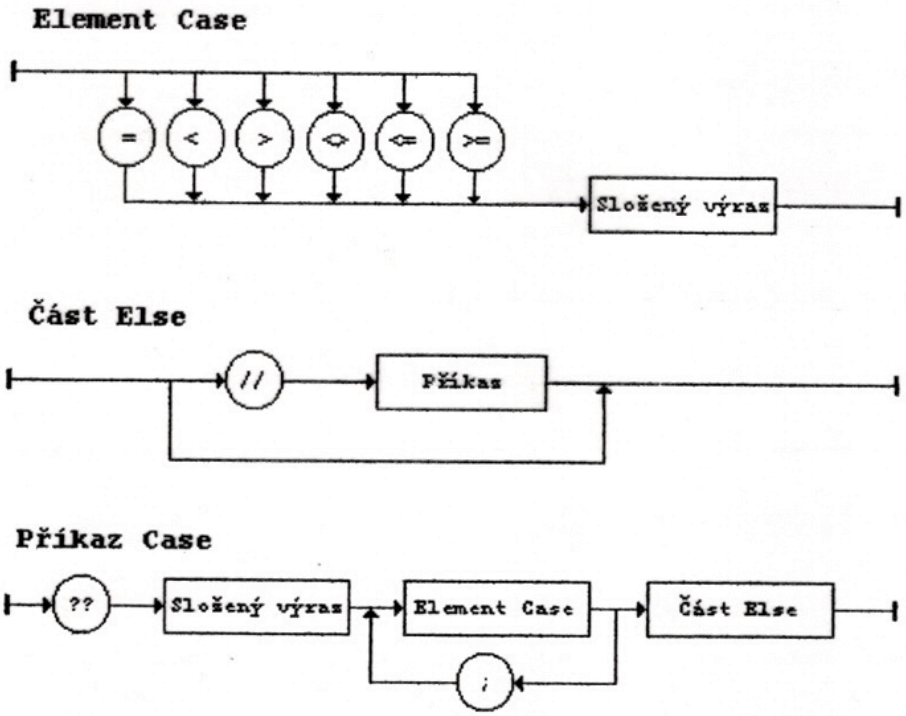
Typ_Struktura = 'Identifikator #p Typ | Identifikator #p Seznam_Typu';

Uplna_Promnena = 'Identifikator_promnen | Identifikator_Konst';

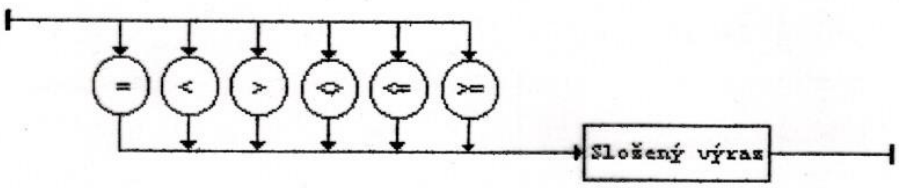
Vyraz = 'Slozeny_Vyraz { Relacni_Operator Slozeny_Vyraz }';

Znamenko = '#+ | #-';

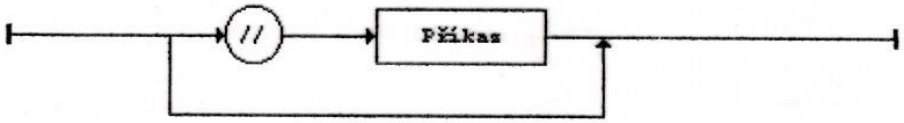
Příloha č.1: Backusova-Naurova forma jazyka Wirth – schéma



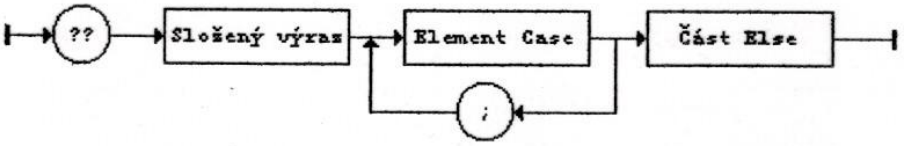
Element Case



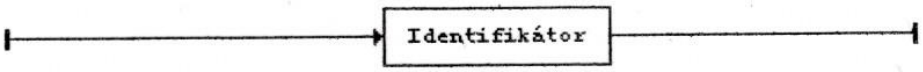
Část Else



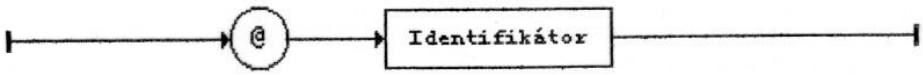
Příkaz Case



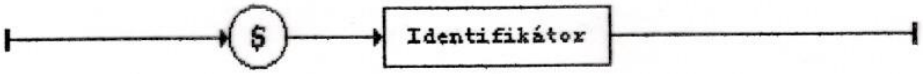
Identifikátor proměnné



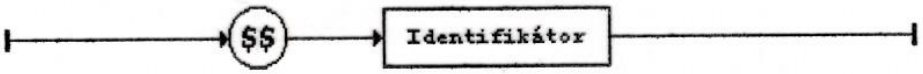
Identifikátor funkce



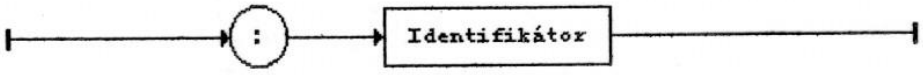
Identifikátor vstupního přerušení



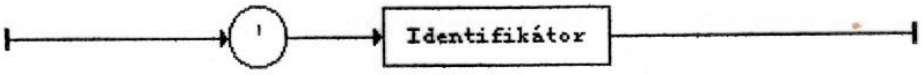
Identifikátor výstupního přerušení



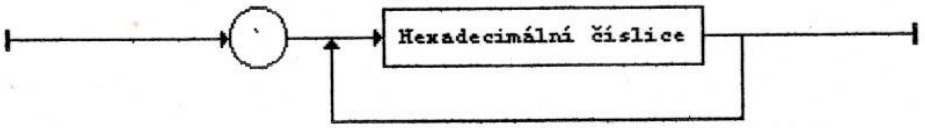
Identifikátor návěští



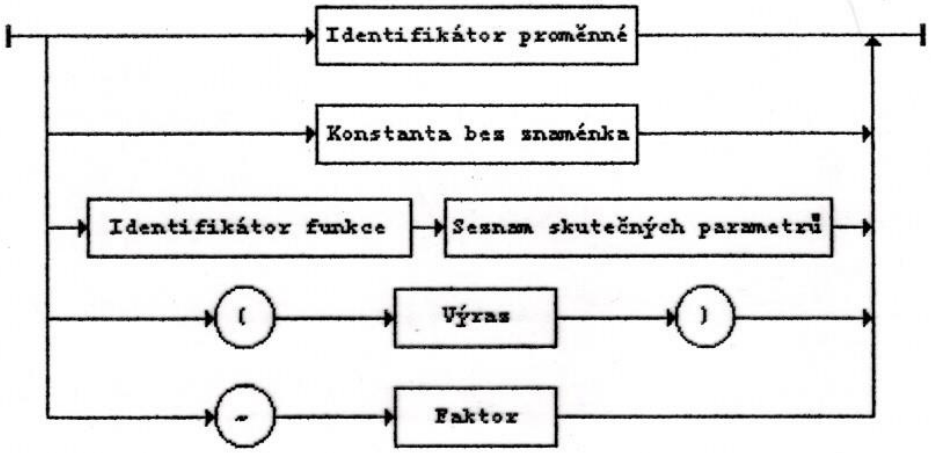
Identifikátor rezervovaného slova



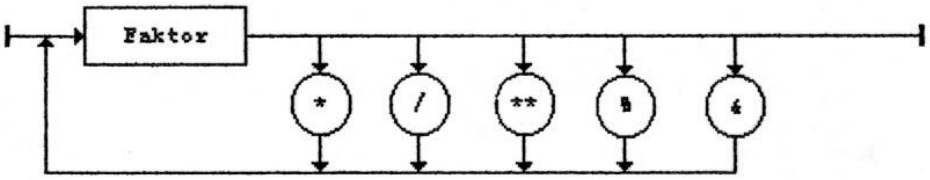
Hexadecimální číslo



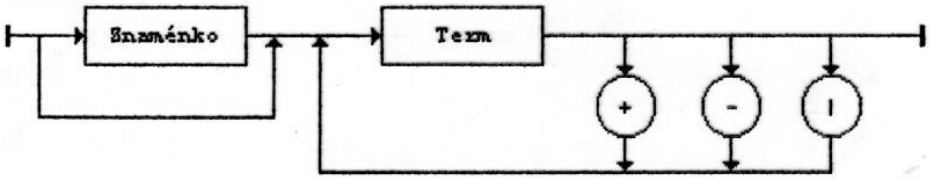
Faktor



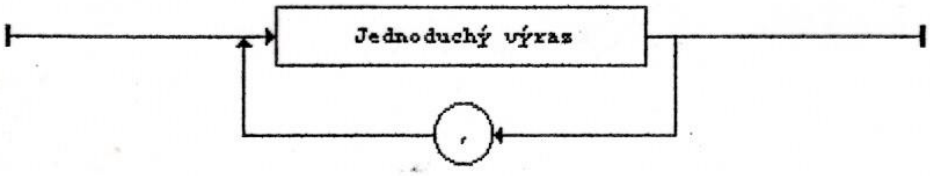
Term



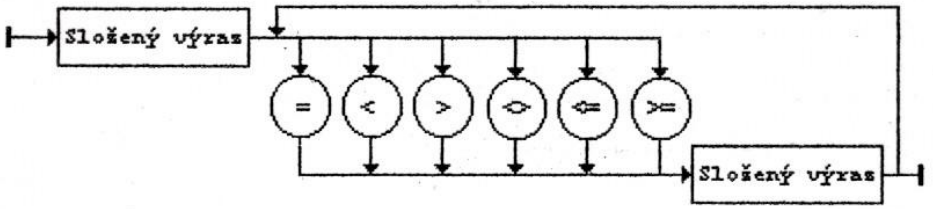
Jednoduchý výraz



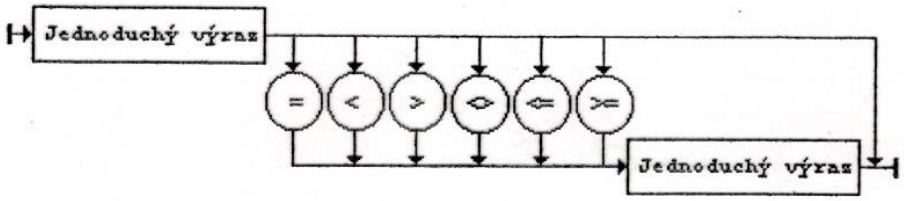
Složený výraz



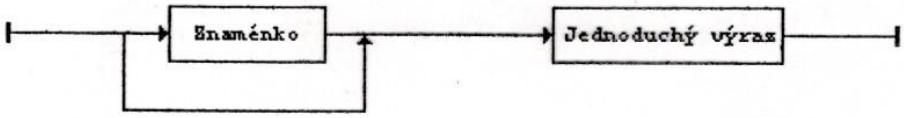
Výraz



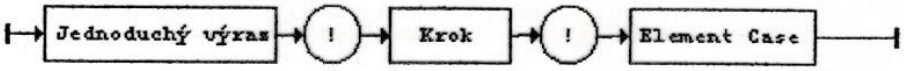
Relační otázka



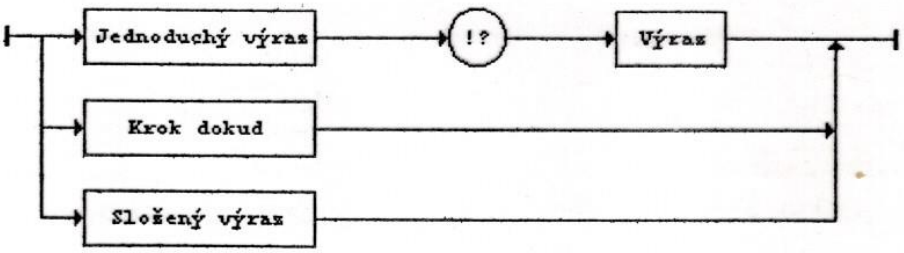
Krok



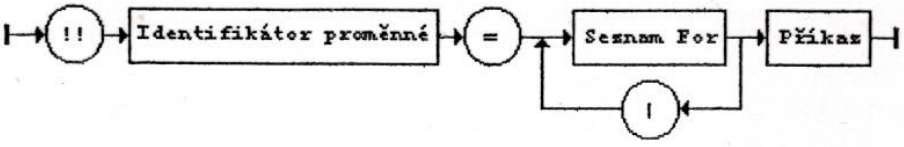
Krok dokud



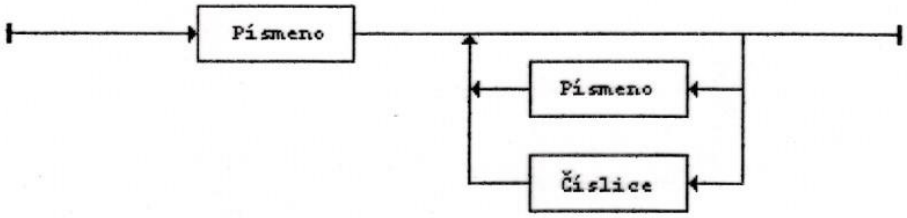
Seznam For



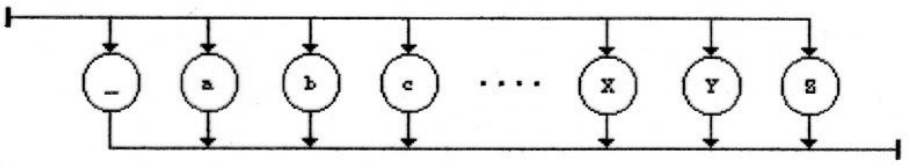
Příkaz For



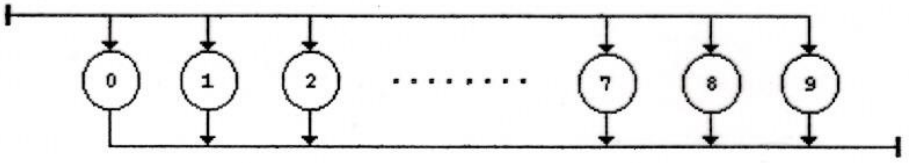
Identifikátor



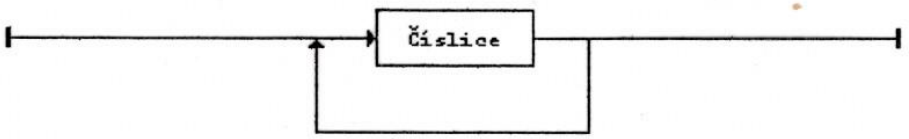
Písmeno



Číslice



Celé číslo bez znaménka



Příloha č.2: Ukázka programu v jazyce ADA

```
-- Priklad programu pro primy pristup ke vzdalenym typum
package Tapes is
          pragma Pure(Tapes);
          type Tape is abstract tagged limited private;
          -- Primitive dispatching operations where
          -- Tape is controlling operand
          procedure Copy (From, To : access Tape; Num_Recs : in Natural) is
abstract;
          procedure Rewind (T : access Tape) is abstract;
          -- More operations
       private
          type Tape is ...
       end Tapes;

       with Tapes;
       package Name_Server is
          pragma Remote_Call_Interface;
          -- Dynamic binding to remote operations is achieved
          -- using the access-to-limited-class-wide type Tape_Ptr
          type Tape_Ptr is access all Tapes.Tape'Class;
          -- The following statically bound remote operations
          -- allow for a name-server capability in this example
          function  Find     (Name : String) return Tape_Ptr;
          procedure Register (Name : in String; T : in Tape_Ptr);
          procedure Remove   (T : in Tape_Ptr);
          -- More operations
       end Name_Server;

       package Tape_Driver is
         -- Declarations are not shown, they are irrelevant here
       end Tape_Driver;

       with Tapes, Name_Server;
       package body Tape_Driver is
          type New_Tape is new Tapes.Tape with ...
          procedure Copy
           (From, To : access New_Tape; Num_Recs: in Natural) is
          begin
            . . .
          end Copy;
          procedure Rewind (T : access New_Tape) is
          begin
             . . .
          end Rewind;
          -- Objects remotely accessible through use
          -- of Name_Server operations
          Tape1, Tape2 : aliased New_Tape;
       begin
          Name_Server.Register ("NINE-TRACK",  Tape1'Access);
          Name_Server.Register ("SEVEN-TRACK", Tape2'Access);
       end Tape_Driver;

       with Tapes, Name_Server;
       -- Tape_Driver is not needed and thus not mentioned in the with_clause
       procedure Tape_Client is
          T1, T2 : Name_Server.Tape_Ptr;
       begin
          T1 := Name_Server.Find ("NINE-TRACK");
```

```
   T2 := Name_Server.Find ("SEVEN-TRACK");
   Tapes.Rewind (T1);
   Tapes.Rewind (T2);
   Tapes.Copy (T1, T2, 3);
end Tape_Client;
```

Příloha č.3: Ukázka programu v jazyce ALGOL

```
Begin comment Řešení soustavy lineárních rovnic iterační metodou;
      real suma, norma, epsilon, novex; integer i,j,n;
      vstup (n); comment n udává počet rovnic;
      begin array b,x[1:n],a[1:n,1:n];
            for i:=1 step 1 until n do
            begin vstup(b[i]);
                  for j:=1 step 1 until n do
                  vstup(a[i,j]); end;
            vstup(epsilon);
      znova:norma:=0.0;
            for i:=1 step 1 until n do
            begin suma:=0.0;
                  for j:=1 step 1 until i-1, i+1 step 1 until n
                  do suma := suma + a[i,j]*x[j];
                  novex := (b[i] - suma)/a[i,j];
                  norma := norma + abs(novex - x[i]);
                  x[i] := novex
            end;
            if norma<epsilon then go to stop else go to znova;
      stop:for i:=1 step 1 until n do
            výstup(x[i])
      end

end konec výpočtu;
```

Příloha č.4: Ukázka programu v jazyce BASIC

```
100 CLS
110 PRINT TAB(34); "PLNENI PLANU'
120 PRINT:PRINT
130 PRINT TAB(12);
140 GOSUB 400 'tisk horni horizont
150 R = (76-12)/(800-ø)
160 FOR K = 1 TO 12
170 READ M$, S
180 P = INT((S-ø)* R + 12 + .5)
190 PRINT M$; TAB(12); " ";
200 FOR C = 13 TO P
210       PRINT "*";
220 NEXT C
230 PRINT TAB(76); " "
240 NEXT K
250 PRINT TAB(12);
260 GOSUB 400 'tisk dolni horizont.
270 PRINT TAB(12);
280 FOR CI = 1 TO 8 'tisk oznaceni na osu
290       PRINT SPC(8);CI;
300 NEXT CI
310 PRINT TAB(54); "OBJEM VE 100.000 KC"
320 DATA "LEDEN", 210, "UNOR", 150, "BREZEN", 99
330  :   "DUBEN", 250, "KVETEN", 183, "CERVEN", 352
340  :   "CERVENEC", 410, "SRPEN", 390, "ZARI", 300
350 DATA "RIJEN", 651, "LISTOPAD", 724, "PROSINEC", 516
:
400 FOR L = 1 TO 8
410       GOSUB 500
420 NEXT L
430 RETURN

500 PRINT " + "
510 FOR J = 1 TO 7
520       PRINT " - ";
530 NEXT J
540 RETURN
550 END
```

Příloha č.5: Ukázka programu v jazyce C

```
/* KBH.C – Test stavu klavesnice */

#include <stdio.h>
#include <fcntl.h>
#include <termios.h>

int kbhit(void)
{
      char c;
      static struct termios new_term;
      static struct termios old_term;
      static int handle;
      static int init = 0;

      fflush(stdout);
      if(init == 0) {
            init = 1;
            if((handle = open("/dev/tty", O_RDWR | O_SYNC)) == NULL) {
                  fprintf(stderr, "Nelze cist z terminalu! \n");
                  exit(1);
            }
            else {
                  tcgetattr(handle, &old_term);
                  new_term = old_term;
                  new_term.c_lflag &= ~ICANON;
                  new_term.c_lflag &= ~ECHO;
                  new_term.c_cc[VMIN] = 0;
                  new_term.c_cc[VTIME] = 1;
            }
      }

      tcsetattr(handle, TCSANOW, &new_term);
      if(read(handle, &c, 1) <= 0) {
            c = 0;
      }
      else {
            ungetc(c, stdin);
      }
      tcsetattr(handle, TCSANOW, &old_term);

      return(c);
}

main()
{
      int c;

      while(1) {
            putchar('<');
            if(kbhit() != 0) {
                  c = getchar();
                  printf("\nStisknuta klaves - '%c' \n", c);
                  if(c == 'Z')
                        break;
            }
            putchar('>');
      }
}
```

Příloha č.6: Ukázka programu v jazyce COBOL

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ZAK-FAK.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE/COMPUTER. NSP-12.
OBJECT-COMPUTER. NSP-12.
INPUT-OUTPUT SECTION.
FILE-CONTROL. SELECT DIEL-HLAS ASSIGN SDS.
     SELECT POD-FAK-1 ASSIGN TO MP3.
DATA DIVISION.
FILE SECTION.
FD DIEL-HLAS; LABEL RECORD IS OMITTED; DATA RECORD IS
     HLASENKA; RECORD CONTAINS 80 CHARACTERS.
01 HLASENKA
     02 MENO    PICTURE X (15).
     02 CISLO   PICTURE 9 (5).
     02 UTVAR   PICTURE X (4).
     02 TYZDEN  PICTURE 99.
     02 ZAKAZKA PICTURE X (6).
     02 ODPR-HOD
          03 NOR-HOD PICTURE 99; OCCURS 6 TIMES.
          03 NAD-HOD PICTURE 99; OCCURS 6 TIMES.
     02 SADZBA  PICTURE 9.
     02 HOD-SAZ PICTURE 99V99.
     02 DOV-CHOR     PICTURE 99.
     02 POZNAM  PICTURE X (17).
FD POD-FAK-1; LABEL RECORDS ARE STANDARD; DATA RECORD
     IS UCET-1; RECORD CONTAINS 26 CHARACTERS.
01 UCET-1.            PICTURES X(26).
WORKING-STORAGE SECTION.
77 K                  PICTURE 9; COMPUTATIONAL.
77 SUCET 1            PICTURE 999; COMPUTATIONAL.
77 SUCET 2            PICTURE 999; COMPUTATIONAL.
01 POM
     02 OSOB-CIS           PICTURE 9(4).
     02 UTV           PICTURE X(4).
     02 ZAKAZ         PICTURE X(6).
     02 HOD-1         PICTURE 99.
     02 HOD-2         PICTURE 99.
     02 FAKT          PICTURE 9(6) V9(2).
PROCEDURE DIVISION.
     OPEN INPUT DIEL-HLAS, OUTPUT POD-FAK-1.
ZAC. READ DIEL-HLAS RECORD; AT END GO TO UZAVER.
     MOVE ZEROS TO SUCET1, SUCET2.
     PERFORM SUHRN VARYING K FROM 1 BY 1 UNTIL K GREATER 6.
     IF(SADZBA = 1) COMPUTE FAKT = (SUCET1 + 1.5 * SUCET2)*HOD-SAZ
     ELSE GO TO RYCHLO.
PRESUN. MOVE OS-CISLO IN HLASENKA TO OSOB-CIS IN POM,
     MOVE UTVAR IN HLASENKA TO UTV IN POM,]
     MOVE ZAKAZKA IN HLASENKA TO ZAKAZ IN POM.
     WRITE UCET-1 FROM POM.
     GO TO ZAC.
UZAVER. CLOSE DIEL-HLAS, POD-FAK-1.
     STOP RUN.
SUHRN. ADD NOR-HOD(K) TO SUCET1
     ADD NAD-HOD(K) TO SUCET2
RYCHLO. COMPUTE SUC2=(SUCET1+1.5*SUCET2)8HOD-SAZ.
     COMPUTE FAKT=SUC+0.5SUC2 GO TO PRESUN.
```

Příloha č.7: Ukázka programu v jazyce FORTRAN

```
      INTEGER*1  HLASKA(50)
      DATA LASTNO /0/
      . . .
100   FORMAT (50A1)
C Zpracovani chyby s cislem IERRNO
      IDIF = IERRNO - LASTNO
      IF (IDIF) 1,2,3
1     REWIND 4
      DO 11 J = 1,IERRNO
           READ (4,100) HLASKA
11    CONTINUE
      GOTO 4
2     READ (4,100) HLASKA
      GOTO 4
3     DO 13 J = 1,IDIF
           READ (4, 100) HLASKA
13    CONTINUE

4     LASTNO = IERRNO
```

Příloha č.8: Ukázka programu v jazyce PASCAL

```
{Program na vykresleni a pohyb symbolu Sotokan}
USES Graph,Crt;

VAR Gd, Gm: Integer;
    Ukazatel:Pointer;
    x,y:integer;
    a,b:boolean;

PROCEDURE Inverse(var co:boolean);
BEGIN
  co:=not co;
END;

BEGIN
  Gd := Detect;
  InitGraph(Gd, Gm, '\bp\bgi');
  if GraphResult <> grOk then Halt(1);

  Circle(GetMaxX div 2,GetMaxY div 2,100);
  Arc((GetMaxX div 2),(GetMaxY div 2)-50,90,270,50);
  Arc((GetMaxX div 2),(GetMaxY div 2)+50,270,90,50);
  Circle((GetMaxX div 2),(GetMaxY div 2)-50,20);
  Circle((GetMaxX div 2),(GetMaxY div 2)+50,20);
  FloodFill((GetMaxX div 2)+40,(GetMaxY div 2),15);
  FloodFill((GetMaxX div 2),(GetMaxY div 2)+50,15);
  New(Ukazatel);
  GetMem(Ukazatel,   ImageSize((GetMaxX   div   2)-101,(GetMaxY   div   2)-101,
     (GetMaxX div 2)+101,(GetMaxY div 2)+101));
  GetImage((GetMaxX   div   2)-101,(GetMaxY   div   2)-101,(GetMaxX   div   2)+101,
     (GetMaxY div 2)+101, Ukazatel^);
  x:=(GetMaxX div 2)-100;
  y:=(GetMaxY div 2)-100;
  a:=true;
  Randomize;
  SetColor(10);
  ClEArDevice;
  REPEAT
    if a then Inc(x) else Dec(x);
    if b then Inc(y) else Dec(y);
    if (x=-100)or(GetMaxX-302=x) then  Inverse(a);
    if (y=-100)or(GetMaxY-302=y) then  Inverse(b);
    PutImage(x+100,y+100, Ukazatel^, NormalPut);
  UNTIL keypressed;
  CloseGraph;
  FreeMem(Ukazatel, ImageSize((GetMaxX div 2)-101,(GetMaxY div 2)-101,
     (GetMaxX div 2)+101,(GetMaxY div 2)+101));
END.
```

Příloha č.9: Ukázka programu v jazyce PROLOG

```
                    /* Program 55 */
domains
     loc = right; middle; left
predicates
     hanoi(integer)
     move(integer, loc, loc, loc)
     inform(loc, loc)
clauses
     hanoi(N) :- move(A, left, middle, right).

     move(1,A,_,C) :- inform(A,C),!.
     move(A,A,B,C) :- N1=N-1,move(N1,A,C,B),inform(A,C),move(N1,B,A,C).

     inform(Loc1,Loc2):-write("\nMove a disc from ", Loc1," to ", Loc2).
```

Příloha č.10: Ukázka programu v jazyce Wirth

```
(*
   Language WIRTH by Jorel Ralf Hunter

   Example program \
   August \ 1998
*)

@Vetsi(a,b,c>>'int)<<'int;
{
  a,b,c >> 'int;
  pi>>'real:=3.1415297;
  ? a > b,c { Vetsi = a } // ? b > c { Vetsi = b } // ( Vetsi = c );
}

$Nejvetsi;
{
  $$Vypis(@Vetsi(362,45,46+145));
  !! penize = 0.1,0.2,0.5,1,2,5,10,20,50,100,200,500,1000,2000,5000
  {
    ?? penize
      <1    @Write("Drobne penize");
      >=100 @Write("Velke penize");
      //    @Write("Normalni penize") ;
    }
    $$Bankovka(penize);

    ?? Strana
       < "Sever"  @Write("S");
       = "Jih"    @Write("J");
       > "Vychod" @Write("V");
       //         @Write("Z");
    }
  }
  hex >> 'int;
  hex=`5E4BC;
}
```

Příloha č.11: Program Niklaus interpretr jazyka Wirth v jazyce Wirth

```
(*
   Priklad Niclaus - ukazkovy program jazyka Wirth

   Program obsahuje vstupni preruseni:
   1) $Analyza

   a funkce:
   1) @GetElement
   2) @Gramatika

   a pracuje s vystupnima prerusenima:
   1) $$Begining
   2) $$Ending

   Vstupni preruseni $Analyza postupne nacita znaky ze vstupu,
   analyzuje elementy jazyka Wirth a generuje jejich symbolicke elementy.
   Funkce GetElement rozpozna dalsi element jazyka Wirth
   a vrati jeho symbolicky element.

*)

(*
   \\ Promnene definovane jako externi data
   pozice,l       >> 'int;
   ch,chh,element >> 'char;
   radek          >> 'string;
*)

@GetElement;
{
  rad,rade >> 'char;
  element=#7;                                        \\ CHYBA Neznamy element
  rad=Radek[pozice];
  (* Prvni znak *)
  ? (" "<=rad<="~")|(rad=#9)
  {
    ? "0"<=rad<="9" element="1"                      \\ CISLICE
    // ? ("A"<=rad<="Z")&("a"<=rad<="z" element="_" \\ PISMENO
    // ? rad=#9 element=" "                          \\ TABULATOR
    // element=rad;
    (* Druhy znak *)
    ?? element
       = ""","#","$","'",")",",",".",";","[","]","^","{","}","~"," ",#7  *>;
       "a".."z","A".."Z","0".."9","_","`"                                 <*
      //
      {
        ? pozice < l
        {
          pozice++;
          Rade=radek[pozice];
          ?? element =
            "!"                                      \\Krok
               ?? RadE
                 "!" element="f";                    \\Cyklus For
                 "?" element="q";                    \\Dokud v For
                 // pozice--;
               };
            "$"                                      \\Vstupni preruseni
               ? RadE="$"  element="o" // pozice--;  \\Vystupni preruseni
               "&"                                   \\AND
               ? RadE="&"  element="a" // pozice--;  \\NAND
            "("                                      \\Leva zavorka
               ? RadE="*"  element="z" // pozice--;  \\Zacatek komentare
            "*"                                      \\Krat
               ?? RadE
                 "*" element="s";                    \\Mocnina
                 ")" element="k";                    \\Konec komentare
                 "?" element="u";                    \\Dokud Until
                 // pozice--;
               };
            "+"                                      \\Plus
               ? RadE="+"  element="i" // pozice--;  \\Inkrementace

            "-"                                      \\Minus
               ? RadE="-"  element="d" // pozice--;  \\Dekrementace
            "."                                      \\Oddelovac tecka
               ? RadE="."  element="y" // pozice--;  \\Interval
            "/"                                      \\Deleni
               ? RadE="/"  element="e" // pozice--;  \\Jinak ELSE
            "{"                                      \\Zacatek bloku BEGIN
```

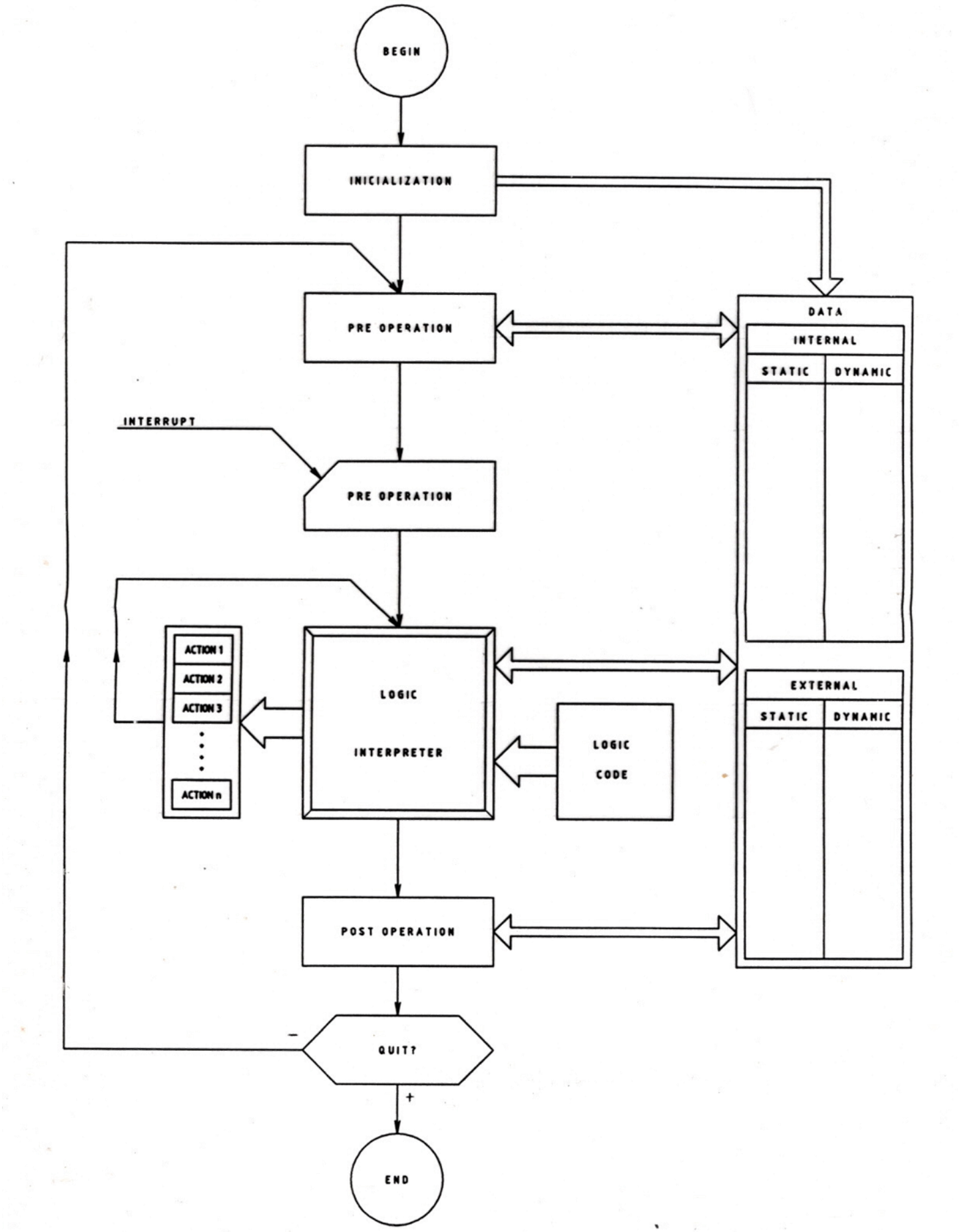
```
            ? RadE="*"  element="r" // pozice--;   \\Zacatek cyklu REPEAT
          ":"                                      \\Navesti
            ?? RadE
              ">" element="j";                     \\Pravda TRUE
              "<" element="b";                     \\Nepravda FALSE
              // pozice--;
            };
          "<"                                      \\Mensi
            ?? RadE
              "<" element="p";                     \\Struktura
              "=" element="m";                     \\Mensi nebo rovno
              ">" element="n";                     \\Nerovno
              // pozice--;
            };
          "="                                      \\Rovno
            ? RadE=">"  element="g" // pozice--;  \\Jdi na GO TO
          ">"                                      \\Vetsi
            ?? RadE
              ">" element="t";                     \\Typ
              "=" element="v";                     \\Vetsi nebo rovno
              // pozice--;
            };
          "?"                                      \\Podminka IF
            ?? RadE
              "?" element="c";                     \\Podminka CASE
              ">" element="w";                     \\Cyklus WHILE
              // pozice--;
            };
          "\"                                      \\Radkovy komentar
            ? RadE<>"\" element=#7 // pozice--;    \\CHYBA Neznamy element
          "|"                                      \\OR
            ?? RadE
              "|" element="r";                     \\NOR
              "&" element="x";                     \\XOR
              // pozice--;
            };
        };
      };
    };
  };
};

@GRAMATIKA;
(*
   Zde by se provedla syntakticka a gramaticka analyza posloupnosti elementu
*)

{
  ; \\ Prazdny prikaz
};

$ANALYZA;
{
  $$Begining
  ?* ~End_of_code
  {
    $$Nacti_radek; \\ Nacte dalsi radek ze zdrojoveho kodu
    l=radek[1];
    ? l>0
    {
      pozice=1;
      {*
        @Get_Element;
        @Gramatika;
        Pozice++;
      *? pozice>l;
    };
  };
  $$Cekej;
  $$Ending;
};
```

Příloha č.12: Tabulka elementů jazyka Wirth

Tabulka elementů jazyka Wirth

Pořadové číslo	ASCII hodnota	Znak	Element	Význam
1	32			Mezera
2	33	!	!	Krok
3	34	"	"	Řetězec
4	35	#	#	Znak
5	36	$	$	Vstupní přerušení
6	37	%	%	Zbytek po dělení
7	38	&	&	Logický AND
8	39	'	'	Rezervované slovo
9	40	(	(	Levá závorka
10	41	)	)	Pravá závorka
11	42	*	*	Krát
12	43	+	+	Plus
13	44	,	,	Oddělovač čárka
14	45	-	-	Mínus
15	46	.	.	Oddělovač tečka
16	47	/	/	Děleno
17	49	1	0..9	Číslice
18	58	:	:	Návěští
19	59	;	;	Konec příkazu
20	60	<	<	Menší
21	61	=	=	Rovno
22	62	>	>	Větší
23	63	?	?	Větvení IF
24	64	@	@	Funkce
25	91	[	[	Začátek pole
26	92	\	\\	Řádkový komentář
27	93	]	]	Konec pole
28	94	^	^	Tabulka typů
29	95	_	_,a..z,A..Z	Písmeno nebo _
30	96	`	`	Hexadecimální číslo
31	97	a	:>	Pravda TRUE
32	98	b	:<	Nepravda FALSE
33	99	c	??	Větvení CASE
34	100	d	--	Dekrement
35	101	e	//	Jinak ELSE
36	102	f	!!	Cyklus FOR
37	103	g	=>	Jdi na GO TO
38	105	i	++	Inkrementace
39	106	j	<*	Přerušení BREAK
40	107	k	*)	Konec komentáře
41	108	l	*>	Pokračuj CONTINUE
42	109	m	<=	Menší nebo rovno
43	110	n	<>	Nerovno
44	111	o	$$	Výstupní přerušení
45	112	p	<<	Struktura
46	113	q	!?	Dokud (v cyklu FOR)
47	114	r	(*	Opakuj REPEAT
48	115	s	**	Mocnina
49	116	t	>>	Typ
50	117	u	*?	Dokud UNTIL
51	118	v	>=	Větší nebo rovno
52	119	w	?*	Pokud WHILE
53	120	x	|&	Logický XOR
54	121	y	..	Interval
55	122	z	(*	Začátek komentáře
56	123	{	{	Začátek bloku
57	124	|	|	Logické OR
58	125	}	}	Konec bloku
59	126	~	~	Logická NEGACE

Příloha č.13: Schéma logického interpreteru (překladače)



NÁVRH PROGRAMOVACÍHO JAZYKA WIRTH

(PREHISTORICKÁ) SEMINÁRNÍ PRÁCE K POCTĚ NIKLAUSE WIRTHA

Autor: **Jiří** (Vyšín) **Altior**

Zadavatel: **SPŠE a** (VTŠ) **VOŠ PARDUBICE**

Datum publikace původního textu: **18. srpna 1998**

Datum publikace revize: **16. září 2022**

Revizi zpracoval: **Jiří Altior**

Pro: **Institut Renesance Kulturního Dědictví, z. ú.**
sekce Aplikované Kreativity (IRKD/AK)

Počet stran: **42**

Forma publikace: **PDF**