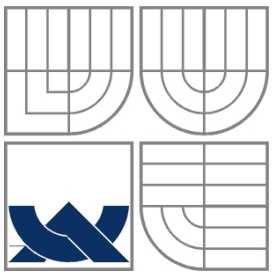


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

ŘÍDICÍ A DATOVÉ STRUKTURY V PROGRAMOVACÍCH JAZYCÍCH

CONTROL AND DATA STRUCTURES IN PROGRAMMING LANGUAGES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAN JANEČEK**
AUTHOR

VEDOUCÍ PRÁCE **RNDR. JIŘÍ DVOŘÁK, CSC.**
SUPERVISOR

BRNO 2010

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav automatizace a informatiky
Akademický rok: 2009/2010

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

student(ka): Jan Janeček

který/která studuje v **bakalářském studijním programu**

obor: **Strojní inženýrství (2301R016)**

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Řídicí a datové struktury v programovacích jazycích

v anglickém jazyce:

Control and data structures in programming languages

Stručná charakteristika problematiky úkolu:

Řídicí struktury (strukturované příkazy) zajišťují organizaci průběhu výpočtu. Patří mezi ně složený příkaz, příkazy větvení a příkazy cyklu. Datové struktury popisují organizaci dat, s nimiž se výpočet provádí. Patří k nim např. pole, množina, záznam a soubor.

Cíle bakalářské práce:

1. Popsat řídicí a datové struktury ve vybraných programovacích jazycích.
2. Uvést příklady jejich použití.
3. Provést jejich srovnání.

Seznam odborné literatury:

WIRTH, N. Algoritmy a štruktúry údajov. Bratislava, ALFA 1988.
HONZÍK, J. a kol. Programovací techniky. Skripta. VUT v Brně, 1985.

Vedoucí bakalářské práce: RNDr. Jiří Dvořák, CSc.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2009/2010.

V Brně, dne 20.11.2009

L.S.

_____	_____
prof. RNDr. Ing. Miloš Šeda, Ph.D.	doc. RNDr. Miroslav Doupovec, CSc.
Ředitel ústavu	Děkan fakulty

ABSTRAKT

Podstatou této práce je uvést základní informace o algoritmizaci a obecné poznatky o řídicích a datových strukturách. Tyto poznatky jsou pak ve vybraných programovacích jazycích konkretizovány, je provedeno jejich srovnání a na základě konkrétních zákonitostí jsou uvedeny příklady využití.

ABSTRACT

The essence of this work is to give basic information about the algorithm development and general pieces of knowledge of the structures in programming languages. These pieces of knowledge are specified for selected programming languages, to comparison is performed and application examples based on specific patterns for corresponding language are presented.

KLÍČOVÁ SLOVA

Řídicí struktury, datové struktury, programování, programovací jazyk

KEYWORDS

Control structures, data structures, programming, programming language

PODĚKOVÁNÍ

Děkují tímto RNDr. Jiřímu Dvořákovi za cenné připomínky a rady při vypracování bakalářské práce.

Obsah:

1 ÚVOD

V dnešní době, kdy stále větší roli v našem životě hrají počítače a stroje obecně, se setkáváme se stále náročnějšími požadavky na tyto stroje. Aby mohl stroj plnit funkci, pro kterou je určen, je třeba krom jeho správné konstrukce, která musí vyhovovat zadaným požadavkům, také informace o tom, jak má dané úkony provádět.

Pro potřeby této práce bude stroj reprezentován počítačem. Počítač budeme chápat jako zařízení, které zpracovává zadanou matematickou úlohu, ať už řešení probíhá jakýmkoliv způsobem. Moderní počítač, jak ho dnes známe, používá při výpočtech jako nosné médium údajů elektrický proud nebo napětí a je tvořen soustavou elektrických obvodů.

Název počítač není náhodný, je odvozen ze slova počítání. V počítačí jsou totiž matematické úlohy řešeny jako posloupnost jednodušších matematických operací.

Charakteristické vlastnosti počítačů lze shrnout takto [1]:

1. Počítače jsou schopné řešit i velmi rozsáhlou úlohu podle dopředu daného předpisu úplně automaticky.
2. Činnost počítačů se vyznačuje velkou rychlostí, proto mohou řešit i velmi rozsáhlé úlohy v poměrně krátkém čase.

Jak už bylo řečeno, počítač řeší zadané matematické úlohy pomocí jednodušších operací, proto abychom mohli využít počítač k řešení problému, je nutné, abychom věděli, jak danou úlohu rozepsat do těchto jednodušších operací. Jednodušší operace jsou primitivní činnosti, které má počítač obsažené ve své instrukční sadě a žádné jednodušší činnosti nemůže použít. Tyto primitivní činnosti odpovídají příkazům procesoru a používají se v nižších programovacích jazycích, které mají složitý zdrojový kód. Vyšší programovací jazyky mají mnohem srozumitelnější kód, jejich převod do strojového kódu, což je nižší programovací jazyk, se provádí pomocí kompilátoru.

Základní myšlenkou programování je na základě určitých informací vytvořit ze stávajících dat data nová. Řešit úlohu tedy znamená transformovat vstupní údaje na výstupní. To, jakým způsobem se vstupní údaje transformují na výstupní, záleží na posloupnosti jednotlivých operací. Správná posloupnost výkonu operací je základní princip programování vedoucí ke správnému výsledku řešeného problému. Abychom dosáhli správného řešení problému, je třeba splnění výstupních podmínek. Zároveň však musí být splněny i vstupní podmínky. Oba typy podmínek nám charakterizují daný problém, který nazveme algoritmický problém [1].

2 ALGORITMUS

Algoritmus může mít pro stejný problém různé, někdy velmi odlišné, podoby. Je třeba pro daný problém vybrat co nejefektivnější algoritmus, který je srozumitelný, jasný a podává výsledky v co možná nejkratším čase .

2.1 Vlastnosti algoritmu

Základní vlastnosti, které musí každý algoritmus mít [2]:

- Hromadnost
  Hromadností rozumíme, že algoritmus je určen pro určitou skupinu úloh, ne jen pro jednu konkrétní. Za vstupní údaje lze tedy použít různé hodnoty, které jsou v určitých mezích řešitelnosti.
- Determinovanost (předurčenost)
  Algoritmus musí být přesný, srozumitelný a jednoznačný. V každém kroku programu musí být jasné, který krok následuje. Při realizaci nesmí být algoritmus ovlivnitelný řešitelem a prostředím, ve kterém se vykonává. Při zadání stejných vstupních dat musí být výsledek identický.
- Resultativnost
  Resultativnost algoritmu znamená, že algoritmus musí skončit po určitém počtu kroků a po tomto počtu kroků musí být známy výsledky algoritmu.
- Efektivnost
  Veškeré operace mají být obecně proveditelné v konečném časovém intervalu pomocí dostupných prostředků.

2.2 Algoritmizace

Algoritmizace, jak už název napovídá, je hledání algoritmu za účelem řešení zadaného problému. I když často hovoříme o hledání algoritmu na řešení daného problému, nejde o hledání ve smyslu procházení existujících algoritmů, ale ve smyslu jeho konstrukce. Třeba hned na začátku říci, že konstrukce algoritmů je tvořivý proces a proto se často hovoří o umění konstruovat algoritmy. To znamená, že neexistuje žádný všeobecný návod na konstrukci algoritmů. Existují však určité rady, získané zkušenostmi, které tento proces mohou ulehčit [1].

Proto při sestavování algoritmu lze postupovat podle určitých etap [3]:

1. Formulace problému
2. Analýza úlohy
3. Vytvoření algoritmu
4. Sestavení programu
5. Odladění programu

Při postupování podle jednotlivých etap nás v první řadě čeká rozbor zadání a to z hlediska vstupních údajů, výstupních údajů, jejich formy a přesnosti řešení. Všechny tyto kroky týkající se zadání lze zahrnout do formulace problému.

Analýza úlohy zahrnuje prvotní rozvahu o řešené úloze. Zjistíme zda je úloha řešitelná, a zda má jeden či více způsobů řešení. Při více možných způsobech bychom si měli zvolit optimální. Z takto zvoleného optimálního způsobu sestavíme sled příkazů, který bude vést k řešení dané úlohy. Sestavení této posloupnosti je 3. etapou algoritmizace.

Na základě sestaveného sledu příkazů neboli algoritmu napíšeme takzvaný zdrojový kód v námi zvoleném programovacím jazyce. Tento kód lze pomocí překladače převést na program. Než bude však program plně funkční, je potřeba provést odladění zdrojového kódu, tedy jeho opravu od logických nebo syntaktických chyb. Na syntaktické chyby nás upozorní překladač a jejich oprava je velmi jednoduchá. Oproti tomu logická chyba je problém přímo ve struktuře kódu a je třeba proto kód přestrukturalizovat. K tomu slouží funkce krokování při odlaďování, kdy se program vykonává v sekvencích a programátor kontroluje funkčnost jednotlivých sekvencí. Popsané úkony v tvoření kódu a ladění spolu úzce souvisí a jsou součástí 4. a 5. etapy, tedy sestavení a odladění programu.

2.3 Zobrazování algoritmu

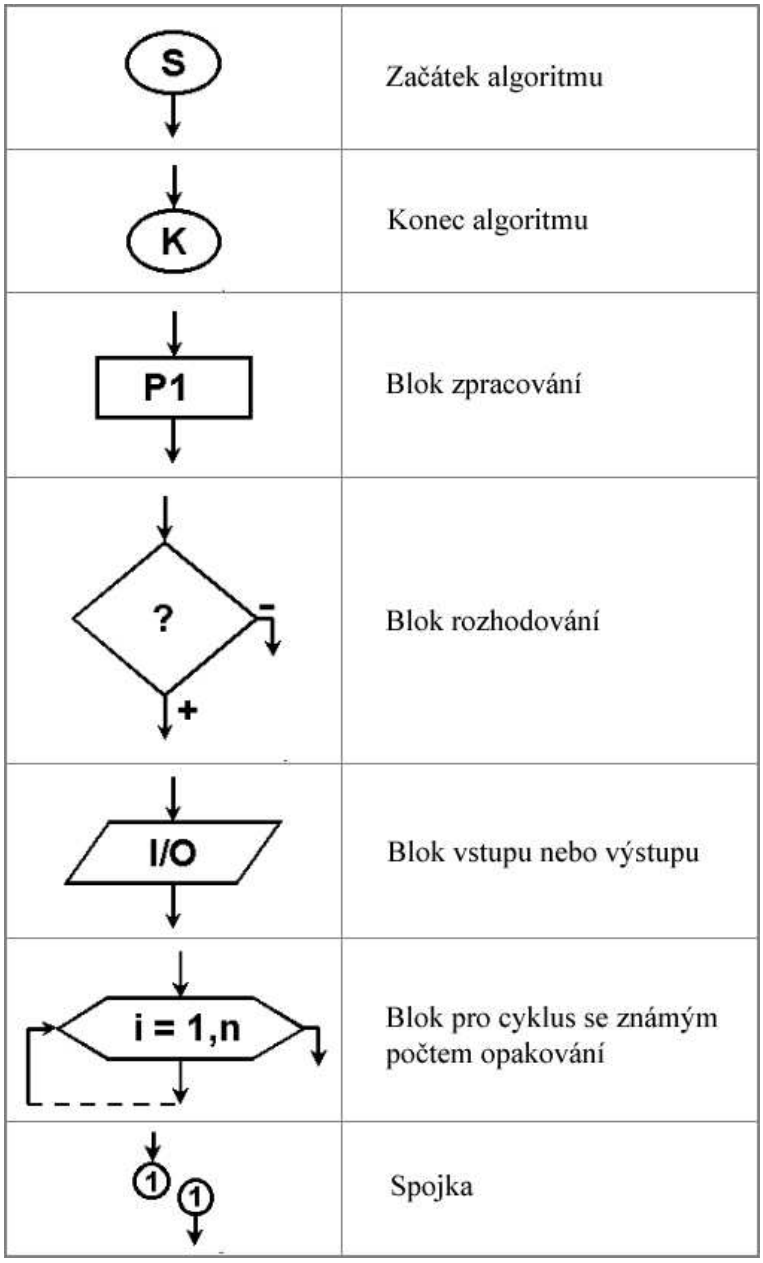
Stejně jako může mít algoritmus mnoho podob, má i více způsobů zobrazování. Každý způsob zobrazování algoritmu má své výhody a nevýhody. Základními způsoby zobrazování algoritmu jsou [2]:

Slovní zápis algoritmu přirozeným jazykem

Slovní zápis algoritmu využívá obecnou znalost jazyka, který je všem známý. Není tedy nutné se učit nic nového a každý může bez obtíží přečíst význam daného algoritmu. Naopak velkou nevýhodou slovního zápisu algoritmu je jeho nejednoznačnost, která je způsobena stavbou jazyka. Přirozený jazyk obsahuje mnoho nadbytečných informací, které dělají zápis složitějším a přitom nemají žádnou výpovědní hodnotu. Tyto nadbytečné informace v zápisu dělají algoritmus málo přehledným.

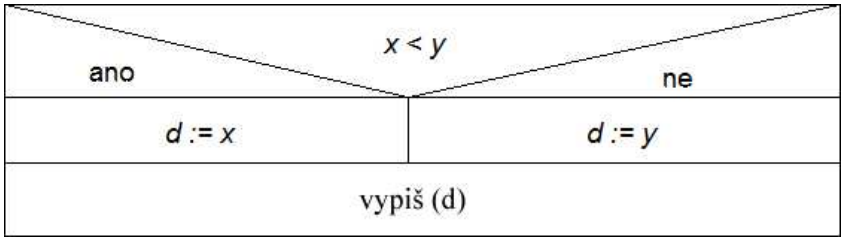
Vývojové diagramy

Vývojové diagramy jsou obrazce zobrazující postup algoritmu grafickou formou. Pro zobrazení jednotlivých úkonů algoritmu se používají definované značky, z nichž každá má přiřazen určitý význam. Přehled jednotlivých značek je na obr. 1. Výhodou vývojových diagramů je jejich grafické 2D zpracování, díky kterému jsou velmi přehledné a názorné. Jejich nevýhoda vyplývá právě z jejich grafické podoby a konstrukce, která zapřičiňuje komplikace s jejich zápisem, zpracováním. Náročné jsou také na údržbu a případné opravy.



Obr. 1 Značky používané ve vývojových diagramech [2].

Mezi grafické prostředky patří také struktogram. Základní vlastností struktogramu je, že jeho symboly a pravidla jejich skládání nedovolují vytvořit jakékoliv nestrukturované spojení. Při konstrování struktogramů se nepoužívají spojovací symboly, jako tomu je u vývojových diagramů, proto struktogram nikdy nepokračuje na jiné stránce. Algoritmus tedy bývá zpracován na více stránkách, mezi kterými je vztah nadřazenosti nebo podřazenosti. Je to méně používaná forma grafického zápisu algoritmu. Ukázka struktogramu je na obr. 2.



Obr. 2 Struktogram.

Speciální jazyky

Pro odstranění zmíněných nevýhod zobrazování algoritmů existuje mnoho speciálních jazyků. Tyto jazyky jsou založeny na určitém přirozeném jazyku, ale používají z něho jen omezenou část slov, kterým přiřazují určitý význam. Takto vybraným slovům říkáme vyhrazená slova. Dalšími prvky speciálního jazyka jsou konstanty a proměnné. Proměnná je objekt algoritmu, který může při vykonávání nabývat různých hodnot. Konstrukce algoritmu ve speciálním jazyce je kombinace vyhrazených slov, konstant a proměnných ve správném pořadí a smyslu.

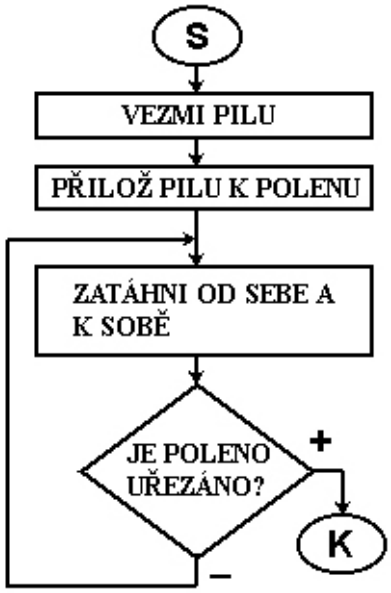
Pro obecné příklady syntaxí jednotlivých příkazů v této práci je použit smyšlený pseudojazyk, který strukturou vychází s jazyka Pascal. Jednotlivá vyhrazená slova, podmínky a příkazy jsou pro lepší porozumění významu popsány pomocí českého jazyka bez použití diakritiky.

Příklad zobrazení algoritmu

Jak bylo uvedeno v předchozí kapitole zobrazení algoritmu má více podob. Zde je příklad využití dvou metod zobrazení a několika etap sestavování algoritmu pro jednoduchý problém [2]:

- formulace problému
  napište algoritmus pro řezání dřeva
- analýza úlohy
  vstupy.....poleno
  výstupy.....uříznutý kus polena
  analýza.....pilou se řeže tak dlouho, dokud není kus polena zcela uříznut
- sestavení algoritmu
  slovní popis:
  1. vezmi pilu
  2. přilož pilu k polenu
  3. zatáhni od sebe a k sobě
  4. je poleno uřezáno?
     ANO – pokračuj bodem 5
     Ne – vrať se na bod 3
  5. ukonči řezání a odlož pilu

  vývojový diagram:



Obr. 3 Vývojový diagram pro jednoduchý algoritmus řezání dřeva.

3 DATOVÉ STRUKTURY

Jak uvádí název knihy od Niklause Wirtha: Algorithms + Data Structures = Programs [14], tak nedílnou součástí programu jsou bezpochyby datové struktury, neboli organizace dat v paměti. I v matematice bývá zvykem důsledně odlišovat jednotlivé proměnné podle svého typu. Programovací jazyky dodržují podobný princip, tedy že každá konstanta, proměnná, výraz nebo funkce je určitého typu [4].

Základním dělením datových struktur je dělení na [5]:

- statické datové struktury
- dynamické datové struktury

Rozdíl mezi dynamickými a statickými datovými strukturami je takový, že dynamické datové struktury mohou v průběhu chodu programu měnit svůj počet složek, tedy velikost přidělené paměti. Naproti tomu statické datové struktury mají pro své složky vymezenu pamět již v průběhu kompilace programu a během chodu už není možné množství paměti měnit.

Další možností dělení datových struktur je podle vlastností jednotlivých složek struktury [5]:

- homogenní – všechny elementy v datové struktuře jsou stejného typu.
- heterogenní – elementy v datové struktuře mohou být různých typů.

Podle složitosti datových typů, lze tyto dělit na jednoduché datové typy a strukturované datové typy.

3.1 Jednoduché datové typy

Všechny datové typy jsou v paměti počítače reprezentovány 0 a 1, rozsah hodnot daného typu čísla je tedy dán vymezeným místem pro uložení 0 a 1. Každý datový typ má vymezeno různé množství místa. U všech jazyků nejsou datové typy totožné a vymezené místo také není stejné. Mezi standardní jednoduché datové typy patří integer, boolean, real a char [6]:

Datový typ Real

Jde o reálná čísla, tedy o nekonečnou množinu čísel. V počítači toto realizovat nelze, protože nemáme k dispozici nekonečnou paměť. Reprezentace reálného čísla v počítači má tedy své omezení a v pravém slova smyslu se tedy nejedná o reálné číslo ale pouze o určitou podmnožinu reálných čísel. Z tohoto důvodu při počítání s reálnými čísly vzniká při výpočtech určitá chyba.Reálné číslo je v paměti počítače uloženo do dvou částí: mantisa, exponent.

Při aritmetických operacích musíme dodržovat takové operandy, které nám zaručí výsledek v požadovaném datovém typu. U datového typu real jsou povoleny 4 základní aritmetické operace: sčítání, odčítání, násobení a dělení. Priorita těchto operací je shodná s prioritou operací v matematice. I za použití pouze povolených operátorů mohou nastat kritické stavy: stav nezobrazitelnosti, stav ztráty hodnoty a stav neřešitelnosti. Stav nezobrazitelnosti neboli stav přeplnění znamená, že výsledek aritmetické operace se nenachází v povoleném rozsahu pro daný datový typ. Stav neřešitelnosti je například dělení nulou, kdy tento výraz nelze aritmeticky vyhodnotit. Posledním kritickým stavem je stav ztráty hodnoty neboli vznik strojové nuly, kdy například při násobení dvou čísel s velkým počtem nul za desetinnou čárkou, ale přesto zobrazitelných, dojde po operaci k tak velkému nárůstu nul za desetinnou čárkou, že na zobrazitelné části desetinného čísla jsou samé nuly a vznikne strojová nula.

Datový typ Integer

Datový typ integer je číselný typ, jeho hodnoty jsou podmnožinou takzvaných celých čísel z matematiky. Povolené aritmetické operace jsou: sčítání, odčítání a násobení. Mezi povolené typy tedy nepatří dělení, je to z důvodu možného nezachování výsledku jako celočíselného datového typu. Z tohoto důvodu jsou u datového typu integer povoleny další dvě aritmetické operace a to: celočíselné dělení (div) a zbytek po dělení (mod). Funkce celočíselného dělení zachovává celočíselný datový typ výsledku díky zaokrouhlování reálného výsledku na celá čísla. Kritickou chybou u tohoto datového typu je nezobrazitelnost, tedy přeplnění, což je stejně jako u předchozího typu přetečení výsledku mimo definovaný rozsah hodnot.

Datový typ Boolean

Tento datový typ už není číslo nýbrž logická hodnota. Může nabývat dvou hodnot: logická pravda (true) nebo logická nepravda (false). Tyto dvě hodnoty jsou jediné přípustné a jsou reprezentovány: logická pravda jako 1 a logická nepravda jako 0. U tohoto datového typu nejsou povoleny žádné aritmetické operace. Povolené operace jsou relační: =, <>, <, >, <=, >= a logické: negace, konjunkce (logický součin) a disjunkce (logický součet). Výsledky logických operací jsou vidět v tabulce 1.

Proměnná		Logické operace		
A	B	negace A	A konjunkce B	A disjunkce B
PRAVDA	PRAVDA	NEPRAVDA	PRAVDA	PRAVDA
PRAVDA	NEPRAVDA	NEPRAVDA	NEPRAVDA	PRAVDA
NEPRAVDA	PRAVDA	PRAVDA	NEPRAVDA	PRAVDA
NEPRAVDA	NEPRAVDA	PRAVDA	NEPRAVDA	NEPRAVDA

Tab. 1 Výsledky logických operací.

Datový typ Char

Je to datový typ, který má jako své hodnoty znaky. Tyto znaky jsou obsaženy v sadě znaků, která obsahuje jak znaky, tak číselnou hodnotu jim odpovídající. Nejčastěji používanou sadou znaků je ASCII (American Standard Code for Information Interchange - „americký standardní kód pro výměnu informací“), další znakovou sadou je například ISO. Zápis hodnoty tohoto datového typu musí vždy být v uvozovkách: 'A', '5' atd. Pro tento datový typ existují 2 funkce zajišťující konverzi mezi znakem a jeho číselným označením ve znakové sadě: **znak**(i) a **číslo**(z), kde **znak** a **číslo** jsou názvy funkcí, i je pořadové číslo ve znakové sadě a z je znak. Obě funkce jsou navzájem inverzní, takže platí **znak**(**číslo**('z')) = z.

3.2 Strukturované datové typy

S ohledem na praktické problémy representace a využití je nutné, aby programovací jazyky měly několik metod strukturování. Z matematického hlediska mohou být všechny ekvivalentní, ale liší se operátory. Mezi základní metody strukturování patří [3],[4]:

- pole
- záznam
- soubor

3.2.1 Pole

Pole je homogenní datová struktura, která se vyznačuje několika důležitými znaky. Všechny jeho složky jsou stejného typu. Jednotlivé složky musí být stejně dosažitelné za stejného okamžiku, to znamená, že ke každé složce mohu přistoupit náhodně. Přístup k jednotlivým složkám je pomocí selektoru s indexem. Za selektor indexu lze dosadit jak přímo hodnotu tak proměnnou. Proměnná se v okamžiku přístupu vyhodnotí a její obsah slouží jako index. Z toho vyplývá velká výhoda polí díky vypočitatelnosti jejich indexu a následného

využití této vlastnosti například pro cykly. Deklarace datového typu pole je:

A: **pole** [1..n] **slozka**: typ_slozky

kde **pole** a **slozka** jsou vyhrazená slova, A je název proměnné typu pole, 1..n jsou indexy pole od 1 do n složek a typ_slozky je datový typ všech složek uvnitř pole.

Přirozeným rozšířením struktury je, aby složky pole byly také strukturované. Docílíme toho tím, že složka pole bude opět typu pole:

A: **pole** [1..n1] **slozka**: **pole** [1..n2] **slozka**: typ_slozky
A: **pole** [1..n1,1..n2] **slozka**: typ_slozky

kde význam všech členů v deklaraci je stejný jako u jednorozměrného pole. Oba zápisy jsou analogické, ale běžněji se používá zjednodušený zápis druhého tvaru.

3.2.2 Záznam

Jde o datový typ, který může být heterogenní. Každá z jeho položek tedy může být jiného datového typu. Každá z položek má své jméno, takzvaný identifikátor položky. Pomocí identifikátoru položky lze ke každé položce přistoupit, nevýhodou identifikátoru položky je, že se nedá vypočítat a nelze ho nahradit jiným objektem, jako to bylo možné u pole. Přístup k dané položce záznamu je tedy přes identifikátor položky, který následuje za selektorem záznamu odděleném tečkou: selektor_záznamu.selektor_položky. Záznamy lze vnořovat do sebe a vytvářet tak rozvětvené struktury. Přiklad deklarace záznamu:

promenna: **zaznam**
    polozka_zaznamu1: datovy_typ;
    polozka_zaznamu2: datovy_typ;

kde **zaznam** je vyhrazené slovo, promenna je název proměnné typu záznam, plozka_zaznamu1,2 jsou položky záznamu definovaných datovych_typu. Pro tento příklad je tedy selektor záznamu: promenna a selektor položky: polozka_zaznamu1 a polozka_zaznamu2. Přístup k položce polozka_zaznamu1 by vypadal: promenna.polozka_zaznamu1, obdobně je to i s položkou polozka_zaznamu2.

3.2.3 Soubor

Jedná se o první datový typ dynamické struktury. Jeho velikost tedy není známa v době překladu, ale za chodu programu se může měnit. Jedná se o homogenní datovou strukturu, tedy všechny její složky jsou stejného typu.

Při deklaraci datového typu soubor je implicitně deklarovaná proměnná používaná na zpřístupnění jednostlivých položek souboru. Proměnná se nazývá ukazatel. Ukazatel slouží k práci se souborem a ukazuje na aktuální položku souboru.

Je zřejmá jistá podobnost s polem, ale na rozdíl od pole nelze přistoupit k jakékoli složce souboru, protože soubor je sekvenční. Tento přístup povoluje pouze postupné procházení složek a čtení té, na které je aktuální pozice ukazatele. Přístup k jednotlivým složkám je tedy realizován pomocí průběžné pozice přístupového mechanismu a jeho pohyb po souboru pomocí operací nad souborem, což bývá obvykle přesun na následující složku nebo na první složku souboru. Důsledkem sekvenčního přístupu k souboru je, že musíme rozlišovat dva stavy souboru, a to stav, kdy se vytváří soubor a lze ho rozšiřovat o další složky, a stav čtení, kdy nelze přidávat žádné nové složky, ale pouze procházet soubor a číst jednotlivé složky. Speciálním typem souboru je textový soubor, který je tvořen posloupností znaků, které jsou organizovány do řádků. Z toho důvodu je pro práci s textovými soubory definován oddělovač řádků a funkce pro práci s ním.

Deklarace datového typu soubor a textový soubor jsou vidět na příkladu:

binarni_soubor: **soubor** typ_slozky
textovy_soubor: **soubor** char

kde **soubor** je vyhrazené slovo, binarni_soubor a textovy_soubor jsou názvy proměnných typu soubor a textový soubor, typ_slozky je deklarace typu složek souboru a char je deklarace složek textového souboru jako znaků.

3.3 Datové struktury v jazyce Pascal

Pro tuto kapitolu bylo čerpáno z [7]. V jazyce Pascal se objevuje datová struktura se, kterou se v ostatních jazycích běžně nesetkáme. Jedná se o množinu.

Deklarace proměnných v jazyce Pascal probíhá v části kódu, před vlastním tělem programu obsahujícím příkazy nebo uvnitř funkcí před blokem příkazů. Deklarační část je uvozena vyhrazeným slovem **var,** za nímž následuje deklarace jednotlivých proměnných, jak je vidět na příkladu:

```
var     promenna1,promenna2,promenna3: datovy_typ;
        promenna4,promenna5: datovy_typ;
        promenna6: datovy_typ;
```

3.3.1 Pole

Deklarace datového typu pole má tuto podobu:

```
var     promenna: array [n1..n2] of datovy_typ;
```

kde **var** je vyhrazené slovo uvozující deklarační část, **array** a **of** jsou vyhrazená slova deklarace pole, promenna je název proměnné typu pole, datovy_typ je datový typ všech složek pole a n1..n2 je rozsah položek pole. Výhodou pole v jazyce Pascal je, že indexování položek nemusí začínat od 0 nebo 1 jako v jiných programovacích jazycích, ale je na programátorovi jakou si zvolí spodní hranici indexu.

Při deklaraci vícerozměrného pole mají prvky identický význam jako pro jednorozměrné pole. Deklarace je jen rozšířena o počet složek pro druhý případně další rozměry pole. Příklad deklarace dvourozměrného pole:

```
var     promenna: array [n1..n2,n3..n4] of datovy_typ;
```

3.3.2 Množina

Základem pro definici datového typu množina musí vždy být ordinální typ.

Ordinální typ je uspořádaná množina hodnot. Každé z těchto hodnot je přiřazeno číslo, které vyjadřuje umístění prvku v množině. Hodnota s číslem neboli ordinalitou jsou spolu úzce spjaty.

Pro množinu jsou povoleny operace:

+	sjednocení dvou množin
-	rozdíl množin
*	průnik dvou množin
=	rovnost dvou množin
<>	nerovnost dvou množin
<=	je podmnožinou
>=	obsahuje

Dalším operátorem, který lze u tohoto datového typu použít je operátor **in**, který testuje, zda je daný prvek obsažen v množině či není. Výsledek této operace je typu boolean. Deklarace datového typu množina je:

```
var     promenna: set of ordinalni_datovy_typ;
```

kde **var** je vyhrazené slovo uvozující deklarační část, **set** a **of** jsou vyhrazená slova deklarace množiny, promenna je název proměnné typu množina a ordinalni_datovy_typ je datový typ hodnot množiny. Příklad použití různých ordinálních typů při deklaraci:

```
var     cislo: set of 1..3;
        znak: set of char;
        barva: set of (cervena,zluta,zelena,modra);
```

kde význam vyhrazených slov je identický jako ve výše uvedené deklaraci datového typu množina a cislo,znak,barva jsou názvy proměnných, 1..3 je ordinální typ proměnné cislo, která je tvořena třemi celočíselnými hodnotami 1, 2, 3, char je ordinální typ proměnné znak, tvořené 26 alfabetickými znaky, (cervena,zluta,zelena,modra) je ordinální typ pro proměnnou barva, která je tvořena výčtem o 4 hodnotách.

3.3.3 Záznam

Deklarace datového typu záznam:

```
var     promenna: record
                      polozka_zaznamu1: datovy_typ;
                      polozka_zaznamu2: datovy_typ;
```

kde **var** je vyhrazené slovo uvozující deklarační část, **record** je vyhrazené slovo pro deklaraci záznamu, promenna je proměnná typu záznam, polozka_zaznamu1 a polozka_zaznamu2 jsou položky záznamu definovaných datových typů. Pro jednotlivé položky záznamu lze použít různé datové typy:

```
var     osoba: record
                      jmeno,prijmeni: array [1..20] of char;
                      pohlavi: (muz,zena);
                      vek: integer;
```

kde význam všech vyhrazených slov je shodný s předchozí deklarací, osoba je název proměnné typu záznam, jmeno a prijmeni jsou položky záznamu deklarované jako pole 20 znaků, pohlavi je položka záznamu deklarovaná výčtem 2 prvků a vek je položka záznamu celočíselného datového typu.

Při zpracování většího počtu položek jednoho záznamu je neefektivní přistupovat ke každé položce záznamu pomocí opakovaného uvádění identifikátoru proměnné typu záznam. Proto je vhodné využít příkazu **with**, který dovolí zkrácení zápisu a také zrychlení přístupu k jednotlivým položkám záznamu. Uvnitř příkazu lze položky záznamu označovat pouze identifikátory položek. S využitím deklarací z předchozího příkladu lze zapsat operace se záznamem takto:

```
osoba.jmeno:='Karel';
osoba.prijmeni:='Novotny';
osoba.pohlavi:=muz;
osoba.vek:=18;
```

Tytéž operace se záznamem za použití příkazu with:

```
with osoba do
    begin
        jmeno:='Karel';
        prijmeni:='Novotny';
        pohlavi:=muz;
        vek:=18;
    end;
```

3.3.4 Soubor

Deklarace datového typu soubor a datového typu textový soubor:

```
var     promenna: file of datovy_typ;
        textova_promenna: file of char;
```

kde **var** je vyhrazené slovo uvozující deklarační část, file a of jsou vyhrazená slova deklarace datového typu soubor, promenna je název proměnné typu soubor, textova_promenna je název proměnné typu textový soubor, datovy_typ je libovolný datový typ, s výjimkou typu soubor.

Pro operace se souborem jsou příkazy rozděleny do dvou částí:

- Proces vytváření souboru
  rewrite(F) – tímto příkazem se vytváří prázdný soubor F. Pokud už soubor existuje, zruší se již existující obsah.
  write(F,prvek) – jedná se o funkci, která přidá hodnotu za právě poslední a stane se novou poslední hodnoutou.
- Proces čtení souboru
  reset(F) – význam příkazu je otevření souboru, tedy nastavení souboru do stavu čtení.
  read(F,prvek) – příkaz slouží k přečtení složky souboru. Přečtenou složkou je vždy další v pořadí.

3.4 Datové struktury v jazyce C

Pro deklaraci proměnných v jazyce C neexistuje žádné vyhrazené slovo. Deklarace se tedy provádí přímo uvedením dané deklarace na místě, na kterém je to povoleno. Pro deklaraci proměnné v jazyce C jsou dovolena dvě místa[8]:

- proměnné je možné definovat mimo (vně) funkce. Proměnná se pak nazývá globální a existují po celou dobu provádění programu.
- Proměnné lze definovat uvnitř těl funkcí, ale vždy na začátku bloku, před prvním příkazem nebo voláním funkce. Tyto proměnné jsou nazývány lokální a existují pouze po dobu provádění příslušné funkce.

3.4.1 Pole

Deklarace datového typu pole:

```
datovy_typ promenna[n];
datovy_typ promenna[n1][n2];
```

kde datovy_typ je datový typ všech položek pole, promenna je název proměnné typu pole, n je počet položek pole pro jednorozměrné pole a n1, n2 jsou počty položek pro dvourozměrné pole.

U pole v jazyce C nelze stanovit dolní hranici pole. Pole tedy vždy začíná položkou s indexem [0] a index poslední položky pole tedy je [n – 1]. Tato vlastnost pole je velice důležitá a je nutné aby ji měl programátor na paměti, protože překladač jazyka C nekontroluje meze polí. Může se tedy stát, že přistupujeme k neexistující položce. Výsledkem je přístup na adresu v paměti, která není poli přidělena a často končí pádem programu.

3.4.2 Struktura

Datový typ struktura je obdoba záznamu v Pascalu a jeho deklarace má 3 různé podoby:

1. vytvoření nového druhu existujícího typu

```
struct promenna{
    datovy_typ1 polozka1;
    datovy_typ2 polozka2;
    datovy_typ3 polozka3;
};
```

kde **struct** je vyhrazené slovo, promenna je název nově vytvořeného druhu již existujícího typu a datovy_typ polozka jsou definice jednotlivých položek struktury. Po vytvoření nového druhu, už lze definovat vlastní proměnnou:

```
struct promenna promenna_typu_struktura;
```

2. definování nového datového typu

```
typedef struct {
    datovy_typ1 polozka1;
    datovy_typ2 polozka2;
    datovy_typ3 polozka3;
} PROMENNA;
```

kde je význam členů identický s předchozím případem. Obsahuje navíc vyhrazené slovo **typedef**, které slouží k vytvoření nového datového typu, a pojmenování nově vytvořeného datového typu PROMENNA. Deklarace samotné proměnné je následující:

```
PROMENNA promenna_typu_struktura;
```

3. kombinace obou předchozích

```
typedef struct promenna{
    datovy_typ1 polozka1;
    datovy_typ1 polozka2;
    datovy_typ1 polozka3;
} VYSLEDEK;
```

V posledním případě se jedná pouze o kombinaci dvou předchozích zápisů a deklarace je shodná s druhým typem definice struktury:

```
PROMENNA promenna_typu_struktura;
```

3.4.3 Soubor

Deklarace datového typu soubor:

```
FILE *promenna;
```

kde **FILE** je vyhrazené slov deklarace proměnné typu soubor, * znamená ukazatel a promenna je název proměnné typu soubor. Pro práci se soubory je v jazyce C určeno několik funkcí [8]:

- fopen() – jedná se o univerzální funkcí na otvírání souborů. Otvírá jak soubory textové tak soubory binární. Při použití funkce fopen() je nutné tuto funkci přiřadit do proměnné typu soubor. Funkce fopen() má dva parametry: prvním z nich je řetězec udávající adresářovou cestu k souboru a jeho název, druhým je tzv. mód pro otvírání souboru. Módy pro otvírání souboru jsou:
  - r – otevření souboru
  - w – otevření souboru pro zápis. Používá se ovšem jen pro vytváření nového souboru. Pokud již soubor existuje, je otevřen, ale všechny jeho

údaje jsou vymazány.
    - a – otevření souboru pro přípis. Takto otevřený soubor je připraven pro zápis. Zápis probíhá na místo ukazatele, který je při otvírání nastaven na konec souboru. Při pokusu o otevření neexistujícího souboru je vytvořen nový prázdný soubor.

  Další modifikace módů pro otvírání souboru jsou: Přidáním znaku b za předchozí módy, tím prohlásíme, že se jedná o binární soubor. Stejně tak lze prohlásit, že se jedná o textový soubor písmenem t. Dalším znakem, který lze kombinovat s módy pro otvírání souboru, je znak +, který říká, že soubor otvíráme jak pro čtení tak pro zápis.
  Důležité pravidlo při otvírání souboru je, že při každém otvírání souboru je nezbytně nutné testovat, zda se otevření podařilo. Opomenutí tohoto testu je zdrojem velmi závažných chyb. Testování probíhá pomocí příkazu větvení, kdy v podmínce voláme funkci fopen() a provedení přiřazení, které nám přiřadí návratovou hodnotu. V případě návratové hodnoty rovné NULL, došlo k chybě při otvírání souboru.
- fclose(f) – funkce sloužící k zavření souboru. Parametrem funkce je proměnná odkazující na soubor. Zavření souboru je nedílnou součástí práce se souborem. Každý soubor je po skončení práce nutné zavřít.
- getc(f) – funkce sloužící k získání znaku ze souboru. Při použití této funkce je nutné přiřadit její hodnotu do proměnné. Parametrem funkce je stejně jako u fclose(f) proměnná odkazující na soubor.
- putc(znak, f) – tato funkce slouží k vkládání znaků do souboru otevřeného pro přípis. Funkce má dva parametry, prvním parametrem je zapisovaný znak a druhým je proměnná označující otevřený soubor.

3.5 Datové struktury v jazyce php

Podkladem pro datové struktury v jazyce php jsou publikace [9],[10]. Deklarace v jazyce php nemá žádné své vyhrazené místo a ni vyhrazené slovo uvozující deklarační část. Proměnné se tedy přímo nedeklarují. Deklarace probíhá ve chvíli přiřazení hodnoty do proměnné. Datový typ proměnné se tedy ve chvíli přiřazení nastaví podle vkládané hodnoty. V jazyce php funguje také automatické přetypování, které zajišťuje konverzi datových typů během číselných operací. Při používání proměnných v jazyce php je nutné vždy před identifikátor proměnné uvést znak $, který říká, že se jedná o proměnnou.

3.5.1 Pole

Z důvodů neexistence přímé deklarace proměnných, existuje několik možností, jak přiřadit do proměnné hodnoty typu pole:

```
$promenna[0] = hodnota1;
$promenna[1] = hodnota2;
$promenna[2] = hodnota3;
```

kde promenna je název proměnné typu pole, [0],[1],[2] jsou indexy položek pole a hodnota jsou hodnoty v jednotlivých položkách. Důležitou vlastností pole v jazyce php je, že není homogenní. Každá položka tedy může obsahovat hodnoty jiného datového typu. Při vytváření pole je použito postupné číslování prvků. U pole v jazyce php však nemusí být prvky číslovány postupně, lze použít libovolná čísla prvků nebo dokonce řetězce. Pro přiřazení hodnot do pole se dá také použít kratší zápis:

```
$promenna = array (hodnota1, hodnota2, hodnota3);
```

U tohoto způsobu vytváření pole je vyhrazené slovo **array**, jinak je význam prvků stejný s předchozím případem. Indexy položek nejsou přímo uvedeny, proto budou položkám přiřazeny

postupně od 0 do počtu položek v poli.

I v jazyce php existují vícerozměrná pole. Tvorba vícerozměrného pole spočívá ve vnořování polí, protože položka pole může obsahovat datový typ pole. Zápis vypadá takto:

```
$promenna = array (array (hodnota1, hodnota2), array (hodnota3, hodnota4));
```

Význam prvků je opět shodný s předešlými. I u vícerozměrného pole jde použít místo číselných indexů řetězce. Zápis takového pole by mohl být takovýto:

```
$promenna = array ('retezec1' => array (hodnota1, hodnota2),
                   'retezec2' => array (hodnota3, hodnota4));
```

Přístup k položce jednoho z vnořených polí by tedy mohl vypadat: $promenna['retezec1'][0] nebo $promenna[0][0]. Zápis obou je ekvivalentní a jeho výsledek by byl hodnota1.

3.5.2 Soubor

Pro práci se soubory jsou v jazyce php různé funkce:

- fopen(nazev_souboru, rezim) – tato funkce slouží k otvírání souborů, nebo k jejich vytváření. Použití funkce závisí na režimu. Režimy platné pro tuto funkci jsou [10]:
  - a – otevření souboru pro přidávání, data se přidávají na konec souboru. Pokud soubor neexistuje, je vytvořen.
  - a+ – otevření souboru pro přidávání a čtení. Data se přidávají na konec souboru. Pokud soubor neexistuje, je vytvořen.
  - r – otevření souboru pouze pro čtení.
  - r+ – otevření souboru pro čtení a zápis, data se zapisují za začátek souboru.
  - w – otevření souboru pouze pro zápis. Dojde k vymazání obsahu souboru a v případě neexistujícího souboru je vytvořen nový.
  - w+ – otevření souboru pro zápis a čtení. Opět dojde k vymazání obsahu a pokud soubor neexistuje je vytvořen nový.

  Při použití funkce fopen je nutné přiřadit její hodnotu do proměnné. S touto proměnnou pak dále pracují některé další funkce.
- fclose(identifikator_souboru) – tato funkce zavírá soubor a používá právě proměnnou souboru k identifikaci souboru, který má být zavřen.
- fread(identifikator_souboru, delka) – funkce slouží ke čtení textového souboru. Jeho parametr delka slouží k určení počtu znaků, které se mají přečíst od začátku souboru.
- fgetc(identifikator_souboru) – tato funkce slouží také ke čtení souboru, ale nečte více znaků za sebou, nýbrž jen jeden znak z aktuální pozice ukazatele v souboru.
- fputs(identifikator_souboru,retezec,delka) – funkce sloužící k zápisu do souboru. Poslední parametr delka je nepovinný. Pokud je nento parametr použit, říká kolik znaků se má z parametru retezec zapsat do souboru.
- fwrite(identifikator_souboru,retezec,delka) – funkce identická s funkcí fputs().

3.6 Datové struktury v jazyce Visual Basic

Pro tuto kapitolu bylo čerpáno z [15],[16]. Deklarace proměnných ve Visual Basicu probíhá pomocí vyhrazeného slova **dim**. Deklarace proměnné je možné provést na libovolném místě v kódu. Deklarace proměnné je ukázána na příkladu:

dim promenna1 **as** datovy_typ1
dim promenna2 **as** datovy_typ2 = hodnota

V prvním případě se jedná o vytvoření prázdné proměnné určitého datového typu. V druhém případě také vytvoříme proměnnou určitého datového typu, ale už né prázdnou, nýbrž do ní přiřadíme hodnotu hodnota. V obou variantách jsou dvě vyhrazená slova **dim** a **as** a názvy proměnných promenna1,2 a jejich datové typy datovy_typ1,2

3.6.1 Pole

Deklarace datového typu pole:

dim promenna1(n1) **as** datovy_typ1
dim promenna2() **as** datovy_typ2 = { hodnota1, hodnota2, hodnota3 }
dim promenna3(n2,n3) **as** datovy_typ3

kde **dim** a **as** jsou vyhrazená slova, promenna1,2,3 jsou názvy proměnných typu pole a datovy_typ1,2,3 jsou datové typy složek v jednotlivých polích. Pro pole v jazyce Visual Basic platí, že prvky jsou číslovány od 0 a musíme mít tedy na paměti, že počet prvků v poli je n + 1. V prvním případě deklarace je vytvořeno prázdné pole s indexem posledního prvku pole n1. Ve druhém poli již není délka pole stanovena zadáním počtu prvků, ale do pole jsou dosazeny přímo hodnoty1,2,3, jejichž počet určuje délku pole. V posledním případě jde o vytvoření dvourozměrného pole o rozměrech $(n2+1) \times (n3+1)$.

3.6.2 Soubor

Pro operace se souborem je vyžadována třída System.IO.File, která obsahuje metody pro práci se souborem. Díky této třídě lze provádět různé operace, jako například: vytváření souboru, čtení ze souboru, zápis do souboru. Základní metody práce s textovým souborem jsou:

- createtext(cesta) – metoda vytvářející textový soubor, jejíž parametrem je cesta k místu vytvoření souboru.
- write(), writeline() – metody sloužící k zapsání znaků do souboru. Metoda writeline přidává znaky na nový řádek, kdežto write na řádek stávající.
- readline() – slouží ke čtení ze souboru.
- close() – metoda sloužící k zavření souboru.

4 ŘÍDICÍ STRUKTURY

Řídicí struktury jsou konstrukce, které rozhodují o provádění dalších částí programu. Řídicí struktury tedy nevykonávají žádnou činnost v podobě matematických operací, ale určují sled prováděných příkazů. Program tedy nemusí být lineární, ale může být rozvětvený nebo se některé z jeho částí mohou opakovat.

4.1 Obecný přehled řídicích struktur

Nelineární běh programu je způsoben různými druhy řídicích struktur. Tyto řídicí struktury lze rozdělit do 3 kategorií[1]:

- Složený příkaz
- Příkazy větvení
- Příkazy cyklu

Řídicí struktury jsou součástí strukturovaného programování a to konkrétně rozkladu problému. Rozklad problému spočívá v rozložení původního problému na několik podproblémů. Mezi základní způsoby rozkladu patří: konjunktivní rozklad (složený příkaz), disjunktivní rozklad (příkaz větvení) a repetiční rozklad (příkaz cyklu). Další zásadou strukturovaného programování je tvorba algoritmu postupem shora dolů, která využívá rozkladu a postupně zjemňuje strukturu na podproblémy. Ve strukturovaném programování je zakázáno používat příkazy skoku, které zesložiťují přehlednost a srozuminetelnost kódu. Nedílnou součástí strukturovaného programování je grafická úprava kódu odsazováním vnořených částí. Bez použití tohoto formátu se rozsáhlé kódy stávají značně nepřehlednými.

K rozkladu problému na podroblémy se nepoužívají pouze řídicí struktury ale i procedury a funkce. Procedura a funkce je posloupnost příkazů, které potřebujeme provádět na různých místech v programu. Po vytvoření procerury nebo funkce ji lze použít kdekoli v další části kódu. Vykonání kódu obsaženého v těle procedury, nebo fukce docílíme pomocí volání. Rozdíl mezi procedurou a funkcí je, že funkce vrací hodnotu. Procedura slouží pouze k provedení definovaných příkazů právě v místě jejího volání.

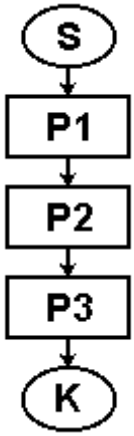
4.1.1 Složený příkaz

Složený příkaz je typ řídicí struktury, která zajišťuje sekvenční plnění příkazů. Jde tedy o příkaz, který zajistí plnění příkazů v takovém pořadí, v jakém jsou uvedeny jednotlivé příkazy v těle složeného příkazu. Složením několika příkazů vytvoříme jeden složený příkaz. Složený příkaz obsahuje vyhrazená slova začátku a konce, uvnitř kterých jsou uvedeny příkazy v pořadí vykonávání. Složený příkaz má tedy tvar:

zacatek_slozeneho_prikazu
    prikaz_P1
    prikaz_P2
    prikaz_P3
konec_slozeneho_prikazu

kde **zacatek_slozeneho_prikazu** a **konec_slozeneho_prikazu** jsou vyhrazená slova a prikazy_P1,P2,P3 jsou jednotlivé příkazy.

Vývojový diagram složeného příkazu je na obr. 4.



Obr. 4 Vývojový diagram složeného příkazu [2].

4.1.2 Příkaz větvení

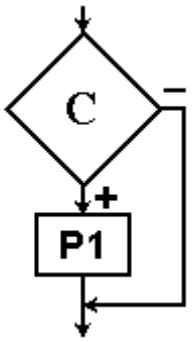
Už z názvu vyplývá, že příkaz větvení rozdělí kód na dvě větve, které se vykonají na základě splnění či nesplnění podmínky. Vždy se vykoná jen jedna větev, nikdy obě. Příkaz větvení lze rozdělit na dva typy:

Příkaz větvení neobsahující příkaz při nesplnění podmínky

Při použití tohoto příkazu větvení je při pravdivém vyhodnocení podmínky splněn příkaz, který je uveden v těle příkazu, ale při nepravdivém vyhodnocení podmínky se žádný příkaz neprovede a příkaz větvení je ukončen. Program dále pokračuje ve vykonávání dalších příkazů. Příkaz větvení tohoto typu má tento tvar:

jestlize podminka_C **potom** prikaz_P1

kde **jestlize** a **potom** jsou vyhrazená slova, podminka_C je vyhodnocovaný výraz s hodnotou pravda nebo nepravda a prikaz_P1 je příkaz vykonaný v případě vyhodnocení podminky_C jako pravda. Slovní vyjádření příkazu tedy je: Pokud je podminka_C pravda, vykoná se prikaz_P1, v případě, že není pravda, nestane se nic. Vývojový diagram tohoto typu příkazu větvení je na obr. 5.



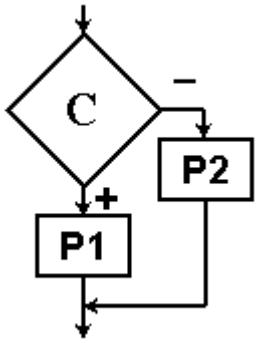
Obr. 5 Vývojový diagram příkazu větvení bez příkazu při nesplnění podmínky [2].

Příkaz větvení obsahující příkaz při nesplnění podmínky

Tento typ příkazu větvení je téměř totožný s příkazem větvení, který neobsahuje příkaz při nesplnění podmínky. Rozdíl spočívá v uvední příkazu pro případ nesplnění podmínky. Výsledkem tedy je, že při nepravdivé podmínce se provede definovaný příkaz namísto ukončení větvení. Zápis příkazu:

jestlize podminka_C **potom** prikaz_P1 **jinak** prikaz_P2

kde **jestlize**, **potom** a **jinak** jsou vyhrazená slova, podminka_C je vyhodnocovaný výraz s hodnotou pravda nebo nepravda, prikaz_P1 je příkaz vykonaný v případě vyhodnocení podminky_C jako pravda a prikaz_P2 se vykoná v případě, že podminka_C je vyhodnocena jako nepravda. Slovní vyjádření příkazu je: Pokud je podminka_C pravda, vykoná se prikaz_P1, v případě, že není pravda, vykoná se prikaz_P2. Vývojový diagram příkazu je na obr. 6.



Obr. 6 Vývojový diagram příkazu větvení s příkazem při nesplnění podmínky [2].

Pomocí příkazu větvení lze vytvářet vnořené struktury, kdy příkaz **jestlize** může v těle obsahovat další příkazy **jestlize**. Použitím této vnořené struktury do větší hloubky se stává kód nepřehledný a rozlišit, do které struktury patří jaký příkaz je složité. Při rozboru vnořených příkazů tedy platí pravidlo: omezovač **else** se vztahuje vždy k nejbližšímu předchozímu omezovači **then**. Rozbor tedy probíhá z nejvnořenější struktury směrem ven.

Příkaz vícenásobného větvení

Speciálním typem větvení je tzv. přepínač. Jedná se o vícenásobné větvení, které obsahuje více jak dvě možnosti pokračování. Příkaz obsahuje selektor, který je porovnáván s definovanými konstantami a na základě shodnosti konstanty a selektoru je proveden příslušný příkaz. Zápis příkazu je takovýto:

prepinac promenna **je**
    konstanta1: prikaz_P1
    konstanta2: prikaz_P2
    konstanta3: prikaz_P3
    jinak:prikaz_P4
konec

kde **prepinac**, **je** a **konec** jsou vyhrazená slova, promenna je selektor porovnávaný s konstantami1,2,3, **jinak** je vyhrazené slovo, za kterým je definován příkaz provedený v případě, že není nalezena shoda selektoru s předchozími konstantami a prikaz_P1,P2,P3,P4 jsou příkazy, které odpovídají jednotlivým konstantám nebo vyhrazenemu slovu **jinak**.

4.1.3 Příkaz cyklu

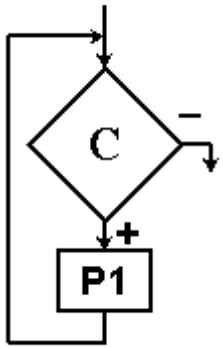
Příkaz cyklu je stejně jako příkaz větvení neodmyslitelnou a důležitou složkou strukturovaného programování. Základním principem příkazu je zajistit několikanásobné opakování určitého příkazu. Počet opakování lze zajistit různými způsoby, z toho důvodu se příkazy cyklu dělí na 3 typy cyklů:

Cyklus s podmínkou na začátku

Principem tohoto cyklu je, že počet opakování je určen dobou, po kterou je splněna podmínka. Není tedy stanoven počet opakování, ale cyklu se provádí tolikrát, kolikrát je splněna podmínka při průchodu. Další charakteristickou vlastností je testování podmínky před vykonáním příkazu. Z toho vyplývá že při nesplnění podmínky hned v prvním průchodu se cyklus ukončí a příkaz se nevykoná ani jednou. Zápis příkazu cyklu:

pokud plati podminka_C **opakuj** prikaz_P1

kde **pokud plati** a **opakuj** jsou vyhrazená slova, podminka_C je výraz testovaný při každém průchodu cyklem a prikaz_P1 je příkaz vykonaný při platné podmínce v každém průchodu cyklu. Slovní vyjádření cyklu je: Pokud je podminka_C pravda opakuj vykonávání prikazu_P1. Vývojový diagram cyklu s podmínkou na začátku je na obr. 7.



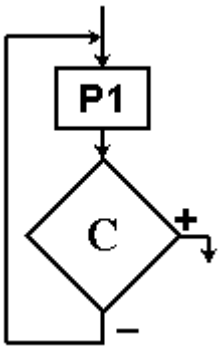
Obr. 7 Vývojový diagram cyklu s podmínkou na začátku [2].

Cyklus s podmínkou na konci

Opět se jedná o cyklus, při kterém se počet opakování udává podmínkou, ale naproti cyklu s podmínkou na začátku se cyklus neopakuje při pravdivé podmínce, nýbrž právě naopak při nesplněné podmínce a právě splněním podmínky cyklus končí. Další odlišností oproti cyklu s podmínkou na začátku je, že příkaz vykonávaný jako opakovaný je před testováním podmínky, tudíž příkaz se provede minimálně jednou i při splněné podmínce na konci cyklu. Zápis cyklu je:

opakuj prikaz_P1 **pokud neplati** podminka_C

kde **opakuj** a **pokud neplati** jsou vyhrazená slova, prikaz_P1 je příkaz opakovaný při běhu cyklu a podminka_C je podmínka zajišťující při nepravdivosti běh cyklu. Slovní vyjádření je: Opakuj prikaz_P1 dokud neplatí podminka_C, jakmile začne platit podminka_C cyklus ukonči. Vývojový diagram k tomuto cyklu je na obr. 8.



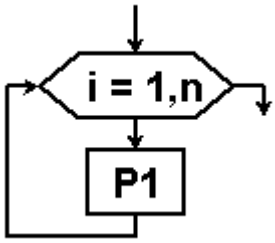
Obr. 8 Vývojový diagram cyklu s podmínkou na konci [2].

Cyklus se známým počtem opakování

Struktura tohoto typu cyklu vypadá na první pohled odlišně, ale je to pouze jiná implementace cyklu s podmínkou na začátku, proto ji nelze považovat za základní iterační strukturu. Její používání je však natolik časté, že je zahrnuta do této práce. V tomto cyklu nefiguruje žádná podmínka, která by určovala počet opakování, ale počet opakování je explicitně zadán přímo v cyklu. Počet opakování lze v cyklu zadat hodnotou celočíselného typu nebo proměnnou celočíselného typu. Zápis cyklu je:

pro i=n1 **do** n2 **opakuj** prikaz_P1

kde **pro**, **do** a **opakuj** jsou vyhrazená slova, i=n1 je proměnná index, označovaná parametr cyklu, s přiřazenou aritmetickou hodnotou n1, n2 je aritmetický výraz celočíselného typu, při jehož dosažení se cyklus ukončí a prikaz_P1 je příkaz opakovaný cyklem. Slovní vyjádření cyklu je: Pro i rovné od n1, které je menší než n2, do n2 prováděj prikaz_P1, přičemž při každém průchodu i zvyš o 1 a při dosažení rovnosti i=n2 cyklus ukonči. Vývojový diagram pro tento typ cyklu je na obr. 9.



Obr. 9 Vývojový diagram cyklu se známým počtem opakování [2].

4.2 Řídicí struktury v jazyce Pascal

Informace pro zpracování řídících struktru v jazyce Pascal jsou čeprány z knihy [7].

4.2.1 Složený příkaz

Syntaxe složeného příkazu:

```
begin
    prikaz_P1;
    prikaz_P2;
    prikaz_P3
end
```

kde **begin** je vyhrazené slovo začátku složeného příkazu, **end** je vyhrazené slovo konce složeného příkazu a prikazy_P1,P2,P3 jsou příkazy v posloupnosti, v jaké se budou postupně provádět. Poslední prikaz_P3 z posloupnosti není ukončen středníkem, protože před **end** není nutné ukončovat příkaz středníkem. Kdybychom středník uvedli, nebylo by to chybné, ale vytvořili bychom zbytečně prázdný příkaz. Složený příkaz zajišťuje nejen vyplnění příkazů v daném pořadí, ale chová se i jako jeden příkaz a lze ho tedy použít v těle ostatních řídicích struktur pro zajištění plnění více příkazů v místech, kde syntaxe povoluje použít pouze jeden příkaz.

4.2.2 Příkaz větvení

Příkaz větvení neobsahující příkaz při nesplnění podmínky

Syntaxe tohoto typu větvení je:

```
if podminka_C then
    prikaz_P1;
```

kde **if** a **then** jsou vyhrazená slova, podminka_C je podmínka, při jejíž splnění se provede prikaz_P1. Chceme-li použít více příkazů, musíme za vyhrazeným slovem **then** vložit složený příkaz, jak je vidět na syntaxi:

```
if podmínka C then
    begin
        prikaz_P1;
        prikaz_P2;
        prikaz_P3
    end;
```

kde celá syntaxe příkazu odpovídá kombinaci složeného příkazu a příkazu větvení.

Příkaz větvení obsahující příkaz při nesplnění podmínky

Syntaxe tohoto typu větvení je:

```
if podminka_C then
    prikaz_P1
else
    prikaz_P2;
```

kde **if**, **then** a **else** jsou vyhrazená slova, podminka_C je podmínka, na základě jejíž logické hodnoty se provádí buď prikaz_P1 při pravdivé (true) podmínce, nebo prikaz_P2 při nepravdivé (false) podmínce. Před vyhrazeným slovem else se příkaz neukončuje středníkem, došlo by tím k vytvoření prázdného příkazu.

Příkaz vícenásobného větvení

V jazyce Pascal je příkaz vícenásobného větvení pojmenován **case** a má syntaxi:

```
case promenna of
    konstanta1: prikaz_P1;
    konstanta2: prikaz_P2;
    konstanta3: prikaz_P3;
    else: prikaz_P4
end
```

kde **case**, **of** a **end** jsou vyhrazená slova pro tvorbu příkazu, **else** je vyhrazené slovo, za kterým následuje příkaz pro případ neshody selektoru promenna s žádnou z konstant a prikaz_P1,P2,P3,P4 jsou příkazy, které se provedou na základě shody s konstantou za kterou se nacházejí.

4.2.3 Příkaz cyklu

Příkaz cyklu s podmínkou na začátku

Jedná se o příkaz cyklu s testováním podmínky ještě před plněním příkazu, tudíž je v syntaxi příkazu nejprve podmínka a až poté vykonávaný příkaz:

```
while podminka_C do
    prikaz_P1;
```

kde **while** a **do** jsou vyhrazená slova, podminka_C je podmínka vyhodnocovaná při každém opakování a dokud je podmínka splněna provádí se prikaz_P1.

Příkaz cyklu s podmínkou na konci

Tento příkaz cyklu testuje podmínku až po vykonání příkazu. V syntaxi je proto nejdříve část pro příkazy a až poté podmínka:

```
repeat
    prikaz_P1;
    prikaz_P2
until podminka_C;
```

kde **repeat** a **until** jsou vyhrazená slova, podminka_C je podmínka, při jejímž splnění se příkaz cyklu ukončí a prikaz_P1,P2 jsou příkazy vykonávané opakovaně. Ze syntaxe vyplývá, že v tomto příkazu cyklu není nutné pro plnění více příkazů používat složený příkaz, protože příkazy jsou zapsány přímo mezi dvěmi vyhrazenými slovy cyklu. Opět není nutné ukončovat poslední příkaz středníkem, protože za ním následuje vyhrazené slovo **until**, které to nevyžaduje.

Příkaz cyklu se známým počtem opakování

Syntaxe cyklu se známým počtem opakování je:

```
for index:=pocatek to konec do
    prikaz_P1;
```

kde **for**, **to** a **do** jsou vyhrazená slova, index je proměnná ordinálního typu, pocatek je počáteční hodnota proměnné index a konec je koncová hodnota proměnné index. Proměnná index se v každém kroku zvyšuje o 1, dokud nedosáhne hodnoty konec, v tom okamžiku cyklus končí. V Pascalu lze využít příkazu cyklu, při kterém se hodnota proměnné index snižuje. Tento cyklus má identickou syntaxi jako cyklus se zvyšující se proměnnou jen místo vyhrazeného slova **to** se použije **downto**. Dalším kriteriem použití cyklu s klesající hodnotou proměnné je, že pocatek musí být větší než konec.

4.3 Řídicí struktury v jazyce C

Hlavním pramenem pro tuto kapitolu je kniha [8].

4.3.1 Složený příkaz

Syntaxe složeného příkazu:

```
{
    prikaz_P1;
    prikaz_P2;
    prikaz_P3;
}
```

kde {} je uvozující a koncová složená závorka ohraničující blok složeného příkazu a prikaz_P1,P2,P3 jsou příkazy vykonávající se v zapsané posloupnosti. Složený příkaz navenek vystupuje jako jediný příkaz a používá se v místech, kde syntaxe jazyka umožňuje pouze jeden příkaz.

4.3.2 Příkaz větvení

Příkaz větvení neobsahující příkaz při nesplnění podmínky

Syntaxe příkazu je následující:

```
if (podminka_C)
    prikaz_P1;
```

kde **if** je vyhrazené slovo příkazu větvení, podminka_C je podmínka rozhodující o provedení prikazu_P1. Při použití příkazu větvení je v těle povolen pouze jeden příkaz, proto je pro více příkazů nutné použít složený příkaz. Syntaxe je shodná, jen jeden příkaz nahradíme příkazem složeným:

```
if (podminka_C) {
        prikaz_P1;
        prikaz_P2;
        prikaz_P3;
}
```

Příkaz větvení obsahující příkaz při nesplnění podmínky

Syntaxe příkazu s else větví:

```
if (podminka_C)
        prikaz_P1;
else
        prikaz_P2;
```

kde **if** je vyhrazené slovo uvozující příkaz větvení, **else** je vyhrazené slovo oddělující příkaz plněný při pravdivé podmínce od příkazu plněného po nepravdivé podmínce, podminka_C je podmínka rozhodující o tom, který příkaz se splní a prikazy_P1,P2 jsou příkazy plněné podle stavu podmínky. Pro nahrazení jednoho příkazu více příkazy je opět nutno použít složeného příkazu. Modifikace je identická s příkazem větvení bez větve **else**. Poslední příkaz před vyhrazeným slovem **else** musí být ukončen středníkem. Toto pravidlo neplatí, pokud je před vyhrazeným slovem **else** konec složeného příkazu, u kterého koncová závorka jasně označuje konec příkazu.

Příkaz vícenásobného větvení

Přepínač v jazyce C se nazývá **switch** a má tuto syntaxi:

```
switch (promenna) {
        case konstanta1: prikaz_P1; break;
        case konstanta2: prikaz_P2; break;
        case konstanta3: prikaz_P3; break;
        default: prikaz_P4; break;
}
```

kde **switch** je vyhrazené slovo pro přepínač, promenna je porovnávaná proměnná neboli selektor, **case** a **break** jsou vyhrazená slova uvozující a ukončující jednotlivé možnosti, **default** je vyhrazené slovo uvozující prikaz_P4 splněný v případě nenalezení dřívější shody mezi selektorem a některou z konstant.

4.3.3 Příkaz cyklu

Příkaz cyklu s podmínkou na začátku

Syntaxe cyklu s podmínkou na začátku je:

```
while (podminka_C)
    prikaz_P1;
```

kde **while** je vyhrazené slovo pro cyklus, podminka_C rozhoduje o provádění nebo neprovádění cyklu a prikaz_P1 se provádí při každém průchodu cyklu a splněné podmínce. Při nesplněné podmínce při prvním průchodu se příkaz cyklu neprovede ani jednou.

Příkaz cyklu s podmínkou na konci

Cyklus s podmínkou na konci má syntaxi:

```
do
    prikaz_P1;
while (podminka_C);
```

kde **do** a **while** jsou vyhrazená slova cyklu, podminka_C je výraz, podle kterého se rozhoduje o opakování cyklu a prikazy_P1,P2 jsou příkazy v těle cyklu. V těle cyklu s podmínkou na konci je dovolen pouze jeden příkaz, proto je nutné použít složený příkaz v případě, že je potřeba více příkazů. Důležitou vlastností je, že cyklus končí při nesplněné podmínce na rozdíl od jazyka Pascal, kde cyklus končí při splněné podmínce.

Příkaz cyklu se známým počtem opakování

Syntaxe cyklu je:

```
for (pocatecni_vyraz; koncovy_vyraz; iteracni_vyraz)
    prikaz_P1;
```

kde **for** je vyhrazené slovo cyklu, pocatecni_vyraz obsahuje přiřazení počáteční hodnoty do řídicí proměnné (např. i = 0), koncovy_vyraz je porovnávací výraz který rozhoduje o ukončení cyklu (např. i < 10) a iteracni_vyraz zajišťuje změnu řídicí proměnné (např. i++ - operátor ++ slouží k zvýšení proměnné i o 1, bez použití delší formy pomocí přiřazovacího příkazu i = i + 1). Jednoduchý příkaz cyklu by tedy mohl vypadat:

```
for (i = 0; i < 10; i++)
    soucet = soucet + i;
```

Za předpokladu, že soucet má před začátkem cyklu nulovou hodnotu, by po skončení cyklu obsahoval součet čísel do 0 do 9. Cyklus v prvním kroku vyhodnotí počáteční výraz a nastaví řídicí proměnnou na 0, v dalším kroku vyhodnotí koncový výraz, a pokud je splněn, což v našem případě je, provede příkaz a vyhodnotí iterační výraz. V dalších krocích již nevyhodnocuje počáteční výraz, ale pouze vyhodnocuje koncový výraz, provádí příkaz v těle cyklu a vyhodnocuje iterační výraz, dokud platí koncový výraz.

4.4 Řídicí struktury v jazyce php

Jazyk php vychází z jazyka C. Díky této skutečnosti jsou základní řídící struktury popsané v této práci pro oba jazyky stejné. Z tohoto důvodu je detailní popis vynechán a syntaxe a vlastnosti pro jazyk php jsou k nalezení v předchozí kapitole Řídicí struktury v jazyce C.

4.5 Řídicí struktury v jazyce Visual Basic

Jako zdroje pro řídící struktury v jazyce Visual Basic jsou použity webové stránky [15].

4.5.1 Příkaz větvení

Příkaz větvení neobsahující příkaz při nesplnění podmínky

Syntaxe příkazu je následující:

```
if podminka_C then
    prikaz_P1
    prikaz_P2
end if
```

kde **if**, **then** a **end if** jsou vyhrazená slova příkazu větvení, podminka_C je podmínka rozhodující o provedení příkazů. V těle příkazu je povoleno uvést více příkazů, protože příkaz obsahuje vyhrazená slova **end if**, která definují konec příkazu větvení.

Příkaz větvení obsahující příkaz při nesplnění podmínky

Syntaxe příkazu s else větví:

```
if podminka_C then
        prikaz_P1
        prikaz_P2
else
        prikaz_P3
        prikaz_P4
end if
```

Význam jednotlivých položek syntaxe je shodný s příkazem větvení bez příkazu při nesplněné podmínce. Obsahuje pouze navíc blok příkazů prikaz_P3,P4, které se provedou práve při nesplněné podmince_C. V obou blocích příkazu je opět povoleno více příkazů.

Příkaz vícenásobného větvení

Přepínač v jazyce Visual Basic se nazývá **select case** a má tuto syntaxi:

```
select case promenna
        case konstanta1
            prikaz_P1
        case konstanta2
            prikaz_P2
        case konstanta3
            prikaz_P3
        case else
            prikaz_P4
end select
```

kde **select case** je vyhrazené slovo začátku přepínače, **end select** je vyhrazené slovo konce přepínače, **case** jsou vyhrazená slova uvozující konstanty a oddělující jednotlivé příkazy a **case else** je vyhrazené slovo uvozující prikaz_P4, který se splní v případě nenalezení shody selektoru promenna s některou z konstant.

4.5.2 Příkaz cyklu

Cykly s podmínkou mohou mít v jazyce Visual Basik 4 podoby. Lze kombinovat podmínku na konci a na začátku s plněním příkazu při splněné nebo nesplněné podmínce.

Syntaxe cyklu s podmínkami na začátku je:

do {while | **until}** podminka_C
    prikazy
loop

kde **do** a **loop** jsou vyhrazená slova cyklu, podminka_C je podmínka rozhodující o počtu opakování příkazů. Při použití cyklu je vždy použito pouze jedno z vyhrazených slov **while** a **until**. **While** znamená opakování cyklu při splněné podmínce a **until** při nesplněné.

Syntaxe cyklu s podmínkami na konci je:

do
    prikazy
loop {while | until} podminka_C

kde **do** a **loop** jsou vyhrazená slova cyklu, podminka_C je podmínka určující počet opakování cyklu a **while** a **until** jsou vyhrazená slova, ze kterých se při zápisu použije jen jedno. Při použití **while** je cyklus opakován při splněné podmínce a při použití **until** při nesplněné. Stále se jedná o příkaz s testováním podmínky až za tělem cyklu, proto první iterace proběhne vždy.

Příkaz cyklu se známým počtem opakování

Syntaxe cyklu je:

for i = n1 **to** n2
    prikaz_P1
    prikaz_P2
    prikaz_P3
next

kde **for, to** a **next** jsou vyhrazená slova, i je řídící proměnná neboli čítač, n1 je počáteční hodnota, n2 je koncová hodnota a prikaz_P1,P2,P3 jsou opakované příkazy. V těle cyklu lze opět použít více příkazů. Počáteční hodnota nemusí být 0 nebo 1, jako tomu je v některých jazycích. Za hodnoty n1 a n2 lze tedy dosadit například 21 a 28. Cyklus se tedy provede 8 krát, protože čítač je zvětšován s krokem 1.

5 APLIKACE ŘÍDICÍCH A DATOVÝCH STRUKTUR

Jedná se o hledání maxima v poli záznamů. Skript je zpracován ve všechy zmíněných jazycích, za využití datové struktury pole, příkazů větvení a cyklu. Skript je ve všech jazycích zpracován co nejvíce podobnou formou, aby bylo možné srovnání. Kód zpracovávající zadaný problém je implementován do funkce max_hodnota, kterou lze zavolat v libovolné části programu, pouze s použitím parametrů představujících vektor údaj obashující informaci o počtu prvků v poli. Při volání funkce je nutné zpracovat hodnotu v návratové proměnné, např. vypsat na obrazovku.

Skript v jazyce Pascal

```
function max_hodnota(var pole: array of integer; pocet_prvku:integer):integer;
        var hodnota,i:integer;
        begin
                for i := 1 to 20 do
                        begin
                                if hodnota <= pole [i] then hodnota := pole [i];
                        end;
                max_hodnota := hodnota
        end;
```

Skript v jazyce C

```
int max_hodnota(int *pole, int pocet_prvku)
        {
          int hodnota;
          int i;
          hodnota = 0;
          for (i = 0; i <= pocet_prvku; i++)
             {
               if (hodnota <= pole [i])
                  {
                     hodnota = pole [i];
                  }
             }
          return hodnota;
        }
```

Skript v jazyce php

```
function max_hodnota($pole, $pocet_prvku)
        {
                $hodnota = 0;
                for ($i = 0; $i <= $pocet_prvku; $i++)
                        {
                                if ($hodnota <= $pole [$i])
                                        {
                                                $hodnota = $pole [$i];
                                        }
                        }
                return $hodnota;
        }
```

Skript v jazyce Visual Basic

```
Function max_hodnota(pole() As Integer, pocet_prvku As Integer) As Integer
    Dim hodnota As Integer
    Dim i As Integer
    hodnota = 0
    For i = 0 To pocet_prvku - 1
      If (hodnota <= pole(i)) Then
        hodnota = pole(i)
      End If
    Next
    max_hodnota = hodnota
End Function
```

6 ZÁVĚR

Cílem práce bylo seznámit čtenáře se strukturami v programovacích jazycích, tedy s datovými a řídícími. Struktury v programovacích jazycích jsou základními kameny programováni a k úspěšnému tvoření je nezbytné, aby se v nich programátor dobře vyznal. S tím ovšem vyvstává problém, že v každém jazyce mají struktury odlišnosti.

Základní vlastnosti struktur, jak jsou popsány v této práci, se téměr nemění, proto v každém jazyce jsou obsaženy různé datové struktury, příkazy větvení a cykly v podobném tvaru, ne-li ve stejném. Bez těchto nástrojů by nebylo možné zkonstruovat rozvětvený program. Avšak drobné odlišnosti v jednotlivých funkcí přeci jen nacházíme.

Každý z jazyků popsaných v této práci se využívá v jiné oblasti programování. Jazyk Pascal, navržený Niklausem Wirthem, byl napsán s cílem vytvořit jazyk vhodný pro výuku programování a navrhnout strukturu tak, aby bylo jazyk snadné implementovat na většině počítačů. První z cílů se daří stále plnit, i po 40 letech existence jazyka. Pascal se stále používá jako výukový programovací jazyk, na kterém se studenti učí chápat programování a získávají základní návyky. Za svůj dlouhý život prošel Pascal mnoha úpravami, největšího rozmachu jazyka však patrně dosáhla implementace Turbo Pascal. S vývojem Turbo Pascalu byla přidána podpora pro objektové programování, která položila základ pro vznik objektově orientovaného jazyka Delphi, který z pascalu vychází [11].

Jazyk C je v současné době jeden z nejpopulárnějších jazyků, používaný jak pro systémové kódy, tak pro psaní aplikací. Jako objektové rozšíření jazyka C byl vytvořen známý jazyk C++, který se stal nejrozšířenějším jazykem pro aplikace Microsoft Windows [12].

Posledním uvedeným jazykem je php. Tento jazyk se nejčastěji používá v souvislosti s tvorbou dynamických internetových stránek. Při tvorbě internetových stránek se často kombinuje s HTML a například databázovým systémem MySQL. Při použití jazyka php pro tvorbu internetových stránek se skript kódu většinou provádí na straně serveru a do prohlížeče na straně klienta je přenesen pouze výsledek, tedy kód například v podobě HTML. Php se stalo oblíbeným díky své jednoduchosti použití a díky určité svobodě v syntaxi, která je důsledkem mírně nekonzistentního vývoje [13].

Ze syntaxe jazyku php je vidět velká podobnost s jazykem C. Důvodem je, že syntaxe jazyka php je inspirovaná jazykem C, ale i jazykem Pascal. Z toho vyplývá souvislost mezi jazyky a důvod podobnosti struktury syntaxu jednotlivých funkcí. Důležitým jazykem, který má souvislost s jazykem C je C#. Byl vyvinut firmou Microsoft a je založen na jazycích C++ a Java, je tedy nepřímým potomkem jazyka C.

Programovací jazyk Visual Basic vychází z jazyka Basic. Oba tyto jazyky byly navrženy, aby bylo snadné se je naučit. Pomocí Visual Basicu jdou vytvářet aplikace s grafickým uživatelským rozhraním. Jedná se jak o jednoduché programy tak programy komplexních struktur. Programování ve Visual Basicu se skládá z vytváření rozhraní pomocí umisťování ovládacích prvků, ke kterým se automaticky píše základní skript a programátor dopisuje skripty které ovlivňují chování jednotlivých prvků a programu.

I když jsou si jazyky podobné, obsahují na první pohled drobné odlišnosti. Pokud by se na ně nebral pří psaní kódu zřetel, mohou při práci způsobit nemalé problémy. Například datový typ pole je v jazyce Pascal a C homogenní strukturou, ale u jazyk php už může být heterogenní strukturou. Stejně tak označování indexů jednotlivých položek pole je u každého jazyka jiné a při přechodu mezi jazyky je důležité si tuto skutečnost uvědomit. Dalším důležitým rozdílem je rozdílné vyhodnocování výrazu u cyklu s podmínkou na konci. Opakování cyklu u jazyka Pascal probíhá, pokud je podmínka nesplněna a končí ve chvíli kdy se podmínka stane pravdou, neboli je splněna. Naproti tomu v C i v php je cyklus opakován právě při splněné podmínce a končí při nesplnění podmínky. Opomenutím této změny vyhodnocování může dojít k zásadním logickým chybám v kódu, které mohou razantně ovlivnit výsledek.

Z uvedených skriptů lze vypozorovat další podobnosti, ale i odlišnosti mezi jednotlivými jazyky. Všechny jazyky dodržují zásady strukturovaného programování, proto je kód ve všech případech dobře čitelný. Nejvíce přehledný je jazyk Pascal, který na rozdíl od

jazyků založených na C používá více vyhrazených slov. V každém jazyku jsou odlišně interpretovány indexy datové struktury pole, proto jsou ve výpočetní části skriptu určité odlišnosti.

S úlohou programátora jde ruku v ruce neustálé studium. Jazyky se stále vyvíjejí, aby uspokojily nové potřeby, které přicházejí s dobou, proto se i programátor musí stále učit. Programátor zpravidla ovládá více jazyků a učí se i nové. Nelze vystačit pouze s jedním jazykem, protože neexistuje žádný univerzální jazyk, který by obsáhl všechny oblasti použití počítačů.

Nejdůležitější znalostí a uměním v programování ovšem stále zůstává algoritmizace, protože ať se jedná o jakýkoli jazyk, stále musí být splněny vstupní i výstupní podmínky, aby bylo dosaženo správného řešení problému.

SEZNAM POUŽITÉ LITERATURY

[1] GVOZDJAK, L. a kol., Počítače a programovanie. Bratislava, ALFA 1985.
[2] DIVIŠ, J.. Algoritmizace [online]. , [cit. 19.5.2010] Dostupné z:<http://www.spsemoh.cz/vyuka/algor/>.
[3] PADRTA, D.. Algoritmy a programování [online]. 1999, 26.3.2001 [cit. 19.5.2010] Dostupné z:<http://david.padrta.sweb.cz/pascal/alg_prog.html>.
[4] HONZÍK, J. a kol., Programovací techniky. Brno, 1985.
[5] Wikipedia contributors. Data structure [online]. 2001, 18. 05. 2010 [cit. 19. 05. 2010] Dostupné z:<http://en.wikipedia.org/w/index.php?title=Data_structure&oldid=362756224>.
[6] Přispěvatelé Wikipedie. Datový typ [online]. 2001, 30. 04. 2010 [cit. 19. 05. 2010] Dostupné z:<http://cs.wikipedia.org/w/index.php?title=Datov %C3%BD_typ&oldid=5286761>.
[7] KVOCH, M., Programování v Turbo Pascalu 7.0. České Budějovice, KOPP 2002.
[8] KADLEC, V., Učíme se programovat v jazyce C. Brno, Computer Press 2005.
[9] WILLIAMS, E.,LANE, D., Programujeme webové aplikace pomocí PHP a MySQL. Praha, Computer Press 2002.
[10] CASTAGNETTO, J. a kol., Programujeme PHP profesionálně. Praha, Computer Press 2001.
[11] Přispěvatelé Wikipedie . Pascal (programovací jazyk) [online]. 2001, 10. 03. 2010 [cit. 23. 05. 2010] Dostupné z:<http://cs.wikipedia.org/w/index.php?title=Pascal_(programovac %C3%AD_jazyk)&oldid=5063979>.
[12] Přispěvatelé Wikipedie . C (programovací jazyk) [online]. 2001, 27. 04. 2010 [cit. 23. 05. 2010] Dostupné z:<http://cs.wikipedia.org/w/index.php?title=C_(programovac %C3%AD_jazyk)&oldid=5273967>.
[13] Přispěvatelé Wikipedie . PHP [online]. 2001, 10. 05. 2010 [cit. 23. 05. 2010] Dostupné z:<http://cs.wikipedia.org/w/index.php?title=PHP&oldid=5326730>.
[14] WIRTH N., Algoritmy a štruktúry údajov. Bratislava, ALFA, 1988.
[15] HERCEG, T., VB.net od začátku [online]. 2007, [cit. 25. 05. 2010] Dostupné z:<http://www.vbnet.cz/>.
[16] WALTON, B., Read and write text files with Visual Basic .NET [online]. 2010, 12. 08. 2002 [cit. 25. 05. 2010] Dostupné z: <http://articles.techrepublic.com.com/5100-10878_11-1045309.html>.