

Ukazatele v jazycích ILE

Vladimír Župka, 2021

Úvod	3
Ukazatele v systému IBM i	4
Ukazatele v jazyku C	4
Ukazatele v jazyku RPG	7
Ukazatele v jazyku Cobol	9
Ukazatele v jazyku CL	11
Příklad v jazyku CL	11
Pomocný program v jazyku RPG	13

Úvod

V tomto článku ukážeme, jak se používají ukazatele v jazycích ILE systému IBM i, tedy v jazycích C, RPG, Cobol a CL. Přitom předvedeme pro každý jazyk příklad se stejnou úlohou, aby se daly prostředky k použití ukazatelů snadněji srovnávat. Vytvoříme pole dekadických čísel o 12 položkách, naplní je samými jedničkami obvyklými prostředky jazyka (s výjimkou CL, kde bude pole vstupním parametrem) a nakonec sečte všechny položky pomocí ukazatelů.

Ukazatele v systému IBM i

Ukazatel (pointer) označuje paměťovou oblast obsahující *adresu* jiné paměťové oblasti nebo procedury. V Systému i zabírá ukazatel v paměti 16 bajtů (128 bitů), z nichž spodních 64 bitů představuje virtuální adresu paměti a v horní polovině jsou uloženy další informace. V systému existuje několik druhů ukazatelů, ale nejdůležitější jsou dva: ukazatele na data a ukazatele na procedury. Zde se budeme zabývat jen ukazateli na data.

Ukazatele v jazyku C

V jazyku C (a také C++) je situace nejsložitější. Jazyk C rozeznává typy ukazatelů podle typů dat.

Ukazatel je proměnná, která obsahuje adresu dat. Lze s ním provádět určité operace: přičíst nebo odečíst celé číslo, spočítat rozdíl dvou ukazatelů, porovnat velikost ukazatelů. Každý ukazatel má stejný typ jako proměnná, na niž ukazuje.

Ukazatel se označuje hvězdičkou v deklaraci:

```
int    *intptr;           // prázdný ukazatel na typ int - neukazuje nikam
char   *ptr;              // prázdný ukazatel na typ char - neukazuje nikam
```

Proměnná *intptr* je ukazatel typu *int*, proměnná *ptr* je ukazatel typu *char*. Stejně deklarace, ale trochu názorněji lze zapsat takto:

```
int*    intptr;
char*    ptr;
```

Hvězdička je zde blíže k typu než ke jménu proměnné toho ukazatele. Ve skutečnosti nezáleží na tom, kolik je mezer mezi jmény a hvězdičkou. Jazyk C dovoluje třeba i zápis bez mezer.

Obsahem ukazatelů v těchto deklaracích je “prázdná adresa” `NULL`.

Ukazatel můžeme v deklaraci inicializovat adresou proměnné:

```
int      cislo = 3;        // proměnná typu int inicializovaná číslem 3
int*     ptr = &cislo;     // ukazatel ptr je inicializován adresou proměnné cislo
```

Chceme-li získat data z proměnné *cislo*, na kterou ukazuje ukazatel *ptr*, použijeme tzv. dereferenci - předřazení hvězdičky před jméno ukazatele:¹

```
int i;                     // proměnná typu int
i = *ptr;                  // bude obsahovat číslo 3
```

Přičtením celého čísla k ukazateli se ukazatel zvýší o *násobek čísla a délky typu*. U odečtení je pravidlo podobné. Přičteme-li tedy k ukazateli číslo 1, zvětší se ukazatel o jedničku jen tehdy, je-li typu *char* (protože typ *char* má délku 1). U typu *int* se přičtením jedničky ukazatel zvětší o 4 (délku typu *int*).²

¹ Samotnému ukazateli se někdy říká reference, získání dat je pak odstranění reference - dereference.

² V době vzniku jazyka C nebylo ještě zcela běžné, že jednotkou adresace je jeden bajt (8 bitů). V některých počítačích to byla “slova” různé bitové délky, (třeba počítač Honeywell 6000 měl 36 nebo 18 bitů podle adresovacího režimu). Celé binární číslo (typ *int* v jazyku C) mělo tedy v různých systémech různou délku. Ve stroji s bajty to byly 4 adresní jednotky (32 bitů jako dnes), zatímco ve stroji se slovy dlouhými 18 bitů to byla 1 adresní jednotka. Aby se program v jazyku C dal použít beze změny v obou systémech, bylo nutné zavést typy ukazatelů a jejich jednotek podle typů dat. Kompilátor v každém systému musel přizpůsobit jednotky ukazatelů jednotkám adres: Přičtením jedničky k ukazateli typu *int* se v “bajtovém” stroji přičte k adrese čtyřka, ve “slovním” stroji se k adrese přičte jen jednička.

```
ptr = ptr + 1;
```

Ukazatel *ptr* bude ukazovat o 4 bajty dále, přičemž nevíme, co na té adrese je. Tato ukázka charakterizuje jazyk C jako náhradu strojových jazyků - assemblerů. Tam se používají aritmetické operace s adresami zcela běžně.

Pole a ukazatele jsou v jazyku C těsně propojeny.

```
decimal(10,2)    array[12];    // prázdné pole s 12 dekadickými položkami
decimal(10,2)*    pointer;      // prázdný ukazatel na typ decimal(10,2)
```

Jméno pole je konstantní ukazatel na první položku, tedy adresa první položky. Proto znamená `array` totéž co `&array[0]`:

```
pointer = array;          /* dosazení adresy začátku pole do ukazatele */
pointer = &array[0];      /* totéž */
```

Počet položek pole zjistíme, když délku pole dělíme délkou první položky. Délky jsou počítány v počtu bajtů:

```
int elements = sizeof(array) / sizeof(array[0]); // počet položek pole
```

Přičteme-li k ukazateli jedničku, zvětší se o délku jedné položky, tj. bude ukazovat na další položku pole. Ukazatel lze porovnávat s ukazatelem stejného typu.

```
for (pointer = &array[0];    // cyklus od první položky
     pointer < &array[0] + elements; // dokud neukazuje za poslední položku
     pointer += 1)           // se zvýšením o velikost jedné položky
    soucet += * pointer;      // přičteme obsah položky (po dereferenci) k součtu
```

Výraz

```
pointer < &array[0] + sizeof(array)/sizeof(array[0])
```

nebo, což je totéž,

```
pointer < array + sizeof(array)/sizeof(array[0])
```

znamená, že porovnáваме ukazatel s adresou, která vznikne, když k adrese první položky přičteme počet položek (12), čili s adresou ukazující těsně za poslední položku.

Následuje ucelený příklad ilustrující souvislost ukazatelů a polí. Zdrojový kód programu má typ C a kompiluje se příkazem `CRTBND C` (volbou 14 v PDM).

```
#include <stdio.h>
#include <decimal.h>

main()
{
    decimal(10,2)    soucet = 0; // deklarace s inicializací
    decimal(10,2)    array[12];  // prázdné pole s dekadickými položkami
    int              i;
    int              elements;    // počet položek pole = 12
    decimal(10,2)    * pointer;   // prázdný ukazatel na typ decimal(10,2)

    i=sizeof (array);
    printf ("Délka pole v bajtech = %d \n", i);

    i = sizeof (decimal(10,2));
    printf ("Délka decimal(10,2) = %d \n", i);

    i = sizeof (array[0]);
    printf ("Délka položky pole = %d \n", i);

    i = sizeof (array) / sizeof(array[0]);
    printf ("Počet položek pole = %d \n", i);
```

```

elements = sizeof (array) / sizeof(array[0]); // počet položek pole

/* Cyklus naplní pole čísly 1 až 12 */
for (i = 0;          // počítá se od nuly do počtu položek - 1!
     i < elements;   // menší než počet položek
     i++ )
    array[i] = i + 1;

/* Cyklus sečte položky pole */
for (pointer = &array[0];    // cyklus od první položky
     pointer < &array[0] + elements; // dokud neukazuje za poslední položku
     pointer += 1)           // se zvýšením o velikost jedné položky
    soucet += * pointer;      // přičteme obsah položky (po dereferenci) k součtu

printf ("Součet = %D(10,2)", soucet);
}

```

Výsledkem spuštění programu je následující výpis na standardním výstupu:

```

Délka pole v bajtech = 72
Délka decimal(10,2) = 6
Délka položky pole = 6
Počet položek pole = 12
Součet = 78.00

```

Aby článek nebyl příliš dlouhý, vynechali jsme další témata jazyka C, jako jsou ukazatele na struktury a přetypování ukazatelů.

Ukazatele v jazyku RPG

Jazyk RPG zná ukazatele na data a ukazatele na procedury. Ukazatele na data jsou proměnné typu *pointer*, které obsahují adresy paměti. Typ *pointer* se označuje v definici hvězdičkou. S ukazateli lze provádět určité operace: přičíst nebo odečíst celé číslo, spočítat rozdíl dvou ukazatelů, porovnat velikost ukazatelů.

Na rozdíl od jazyka C neexistují různé typy ukazatelů. Přičtení celého čísla změní obsah ukazatele (adresu) vždy o toto číslo (počet bajtů) bez ohledu na to, na jaký typ dat ukazuje. Rozdíl ukazatelů dá počet bajtů.

Další rozdíl od jazyka C je ten, že neexistuje dereference ukazatele. Ukazatel můžeme používat jen nepřímo pomocí tzv. *bázované proměnné*. To je v našem příkladu proměnná nazvaná *polozka* označená slovem *based* s uvedením ukazatele *pointer*. Proměnná položka leží vždy na té adrese (bázi), kterou obsahuje ukazatel *pointer*.

Nejprve tedy dosadíme do ukazatele *pointer* adresu první položky pole *array*, v dalších krocích pak získáme adresu další položky přičtením její délky k ukazateli. Proměnná položka vždy obsahuje data z té položky pole, na niž právě ukazuje ukazatel *pointer*.³

Následující ukázka ilustruje stejnou úlohu jako jsme uvedli pro jazyk C. Zdrojový kód programu má typ *RPGLE* a kompiluje se příkazem *CRTBND RPG* (volbou 14 v PDM).

```
**FREE
dcl-s soucet      packed(10: 2);
dcl-s array       packed(10: 2) dim(12);

// počet prvků pole určím při kompilaci funkcí %elem
dcl-s elements    int(10) inz(%elem(array));
dcl-s pointer     POINTER;

// položka je proměnná s vlastnostmi shodnými s položkou pole
// a je bázovaná ukazatelem pointer; zatím nemá přidělenou paměť
dcl-s polozka     like(array) BASED(pointer);
dcl-s idx         int(10);

dsply ('Délka pole v bajtech = ' + %char(%size(array: *all)));
dsply ('Délka decimal (10, 2) v bajtech = ' + %char(%size(soucet)));
dsply ('Délka položky pole v číslicích = ' + %char(%len(array(1))));
dsply ('Počet položek pole = ' + %char(%elem(array)));

// Cyklus naplní pole čísly 1 až 12
for idx = 1 to elements; // cyklus od 1 do počtu položek
  array(idx) = idx;
endfor;

pointer = %ADDR(array); // bázový ukazatel naplním adresou pole

// cyklus sečte položky pole
dow pointer < %addr(array(1)) + %size(array: *all);
  // dokud neukazuje za poslední položku
  soucet += polozka; // přičtu obsah položky bázované ukazatelem
  pointer += %div(%size(array: *all) : elements);
  // a posunu položku o její délku
enddo;

dsply ('Součet = ' + %char(soucet));
return;
```

³ Tento způsob použití ukazatelů je převzat z jazyka PL/I.

Výsledky výpočtu se postupně zobrazují (vždy po stisknutí klávesy Enter) na obrazovce:

```
DSPLY  Délka pole v bajtech = 72
DSPLY  Délka decimal (10, 2) v bajtech = 6
DSPLY  Délka položky pole v číslicích = 10
DSPLY  Počet položek pole = 12
DSPLY  Součet = 78.00
```

Délku položky pole jsme vypočítali v počtu dekadických číslic pomocí funkce `%len`, protože funkci `%size` (délka v počtu bajtů) nelze aplikovat na položku pole.

Výraz

```
%div(%size(array: *all) : elements)
```

představuje délku jedné položky v bajtech. `%div` je funkce celočíselného dělení: velikost celého pole v bajtech se dělí počtem položek a zajišťuje, že výsledek je celé číslo. Kdybychom použili normální operátor `/` pro dělení, kompilátor by hlásil chybu, protože by nebylo obecně zajištěno, že po dělení vyjde celé číslo.

Ukazatele v jazyku Cobol

Jazyk Cobol zná, podobně jako RPG, ukazatele na data a ukazatele na procedury. Ukazatel na data je definován frází `USAGE IS POINTER`.

Ukazatel naplníme funkcí `ADDRESS OF`

```
SET ukazatel TO ADDRESS OF pole.
```

Proměnnou *polozka* definujeme ve spojovací sekci `LINKAGE SECTION` a bázujeme ji stejnou funkcí, ale v obráceném pořadí:

```
SET ADDRESS OF polozka TO ukazatel.
```

Pro ukazatel teď nemůžeme použít jméno `pointer`, protože je to rezervované slovo jazyka Cobol.

Hodnotu ukazatele můžeme zvýšit operací `UP BY` nebo snížit operací `DOWN BY`.

```
SET ukazatel UP BY LENGTH OF polozka
```

S ukazatelem nelze provádět žádné aritmetické operace. Ukazatele lze porovnávat, ale jen na rovnost nebo nerovnost.

```
PERFORM UNTIL ukazatel EQUAL TO konec-pole
```

Následující ukázka ilustruje stejnou úlohu jako jsme uvedli pro jazyk C a RPG. Zdrojový kód programu má typ `CBLLE` a kompiluje se příkazem `CRTBNDCBL` (volbou 14 v PDM).

```
WORKING-STORAGE SECTION.
```

```
01 soucet      PICTURE S9(8)V9(2) PACKED-DECIMAL VALUE 0.
01 i           PICTURE S9(9)  BINARY.
01 elements    PICTURE S9(9)  BINARY.
01 ukazatel    POINTER.
01 konec-pole  POINTER.
01 pole.
05 array OCCURS 12 PICTURE S9(8)V9(2) PACKED-DECIMAL.
```

```
LINKAGE SECTION.
```

```
* tato proměnná nemá přidělenou paměť, bude bázována
* ukazatelem:
01 polozka LIKE array OF pole.
```

```
PROCEDURE DIVISION.
```

```
    DISPLAY "Délka pole v bajtech = " length of pole.
    DISPLAY "Délka proměnné součet = " length of soucet.
    DISPLAY "Délka položky pole = " length of array(1).
    COMPUTE elements = LENGTH OF pole / LENGTH OF array(1).
    DISPLAY "Počet položek pole = " elements.
```

```
    MOVE 1 TO i.
```

```
* cyklus naplní pole čísly od 1 do 12
    PERFORM elements TIMES
        MOVE i TO array(i)
        ADD 1 TO i
    END-PERFORM.
```

```
* ukazatel se nastaví na začátek pole
    SET ukazatel TO ADDRESS OF pole.
* položka dostane adresu (bázi) z ukazatele
    SET ADDRESS OF polozka TO ukazatel.
* konec-pole bude ukazatel zvýšený o délku pole
    SET konec-pole TO ADDRESS OF pole.
    SET konec-pole UP BY LENGTH OF pole.
```

```

* cyklus sečte položky pole
  PERFORM UNTIL ukazatel EQUAL TO konec-pole
* .obsah položky se přičte k součtu, na položku ukazuje ukazatel
  ADD polozka TO soucet
* .ukazatel se zvýší o délku položky, bude ukazovat na další
  SET ukazatel UP BY LENGTH OF polozka
* .položka dostane adresu (bázi) z ukazatele
  SET ADDRESS OF polozka TO ukazatel
  END-PERFORM.

  DISPLAY "Součet = " soucet.

  STOP RUN.

```

Výsledky výpočtu se postupně zobrazují (vždy po stisknutí klávesy Enter) na obrazovce:

```

Délka pole v bajtech = 000000072
Délka proměnné součet = 000000006
Délka položky pole = 000000006
Počet položek pole = 000000012
Součet = 0000007800

```

Součet se vypisuje s dvěma desetinnými místy, ale bez tečky.

Ukazatele v jazyku CL

Práce s ukazateli byla do jazyka CL zavedena ve verzi V5R4. Existuje ovšem jen ukazatel na data.

Příklad se bude od předchozích poněkud lišit, protože jazyk CL neumožňuje práci s poli. Pole tedy nahradíme 72bajtovou znakovou proměnnou, kterou CL program dostane jako vstupní parametr. Tato znaková proměnná obsahuje 12 6bajtových položek představujících dekadická pakovaná čísla o velikosti 10 číslic, z čehož 2 poslední jsou desetinná místa. Přípravu parametru a vyvolání CL programu zajistí pomocný RPG program.

Ukazatel se deklaruje jako proměnná typu *PTR:

```
DCL          VAR(&UKAZATEL) TYPE(*PTR)
```

Lze doplnit parametr ADDRESS, který do něj dosadí adresu dané proměnné:

```
DCL          VAR(&UKAZATEL) TYPE(*PTR) ADDRESS(&ARRAY)
```

Při výpočtu dostane ukazatel hodnotu pomocí funkce %ADDRESS:

```
CHGVAR      VAR(&UKAZATEL) VALUE(%ADDRESS(&ARRAY))
```

Položku pole bázujeme ukazatelem obdobně jako v jazyku RPG:

```
DCL  VAR(&POLOZKA) TYPE(*DEC) STG(*BASED) LEN(10 2) BASPTR(&UKAZATEL)
```

Abychom mohli měnit hodnotu ukazatele, musíme použít funkci %OFFSET, která z ukazatele vyjme *relativní adresu* paměti a uloží ji do proměnné &RELADR typu *UINT (binární číslo bez znaménka dlouhé 4 bajty):⁴

```
DCL          VAR(&RELADR) TYPE(*UINT)
```

```
...
```

```
CHGVAR      VAR(&RELADR) VALUE(%OFFSET(&UKAZATEL))
```

K relativní adrese můžeme přičíst celé číslo nebo je od ní odečíst. Změněnou relativní adresu pak můžeme opět pomocí funkce %OFFSET uložit zpět do ukazatele a změnit tak adresu bázované proměnné.

Příklad v jazyku CL

Následující program je nazván PTRCL a ilustruje použití ukazatele k sečtení položek pole. Zdrojový kód programu má typ CLLE a kompiluje se příkazem CRTBNDCL (volbou 14 v PDM). Je nutné jej vyvolat prostřednictvím programu PTRCL_RPG uvedeného dále.

```
PGM          PARM(&PARAMETR &SOUCET)

/* první parametr je znaková proměnná složená z 12 6bajtových položek */
/* představujících dekadická pakovaná čísla velikosti (10 2) */
DCL          VAR(&PARAMETR) TYPE(*CHAR) LEN(72)
/* druhý parametr je výstupní dekadická proměnná pro součet položek */
DCL          VAR(&SOUCET) TYPE(*DEC) LEN(10 2)

/* lokální kopie parametru */
DCL          VAR(&ARRAY) TYPE(*CHAR) LEN(72)
/* ukazatel na položku pole */
DCL          VAR(&UKAZATEL) TYPE(*PTR) ADDRESS(&ARRAY)
/* položka pole bázovaná ukazatelem */
DCL          VAR(&POLOZKA) TYPE(*DEC) STG(*BASED) LEN(10 2) +
BASPTR(&UKAZATEL)
```

⁴ Relativní adresa se vztahuje k začátku automatické paměti právě probíhajícího vlákna (thread).

```

/* relativní adresa odpovídající ukazateli */
      DCL          VAR(&RELADR) TYPE(*UINT)
/* relativní adresa konce pole */
      DCL          VAR(&KONEC) TYPE(*UINT)

/* kopíruji parametr do pracovní proměnné, protože parametr */
/* je již bazován systémovým ukazatelem */
      CHGVAR      VAR(&ARRAY) VALUE(&PARAMETR)

/* dosadím adresu proměnné array do ukazatele */
      CHGVAR      VAR(&UKAZATEL) VALUE(%ADDRESS(&ARRAY))

/* z ukazatele vyjmu relativní adresu proměnné array */
      CHGVAR      VAR(&RELADR) VALUE(%OFFSET(&UKAZATEL))

/* určím relativní adresu ukazující za konec proměnné array */
      CHGVAR      VAR(&KONEC) VALUE(&RELADR + 72)
/* vynuluji součet */
      CHGVAR      VAR(&SOUCET) VALUE(0)

/* cyklus sečte položky */
      DOWHILE     COND(&RELADR < &KONEC)
      CHGVAR      VAR(&SOUCET) VALUE(&SOUCET + &POLOZKA)

/* .zvýším ukazatel o délku položky v bajtech */
      CHGVAR      VAR(&RELADR) VALUE(&RELADR + 6)
      CHGVAR      VAR(%OFFSET(&UKAZATEL)) VALUE(&RELADR)

      ENDDO

/* vypíšu výsledky do výstupní fronty */
      DMPCLPGM

```

V jazyku CL neexistují funkce pro zjišťování délky proměnných a počtu položek, proto jsou v programu použity konstanty a nevypisují se do výsledků. Příkaz DMPCLPGM vypíše obsah paměti na konci výpočtu, kdy položka je bazována adresou ukazující těsně za konec pole. Proto je její obsah neurčitý.

Pomocný program v jazyku RPG

Následující RPG program PTRCL_RPG vytvoří pole zařazené do datové struktury a naplní je čísly 1 až 12. Datovou strukturu pak předá jako znakový parametr délky 72 bajtů programu PTRCL. Zdrojový kód programu má typ RPGLE a kompiluje se příkazem CRTBNDRPG (volbou 14 v PDM).

```
**free
// datová struktura s polem array
dcl-ds parametr;
    array packed(10: 2) dim(12);
end-ds;

dcl-s i zoned(10);
dcl-s soucet packed(10: 2) inz(0);

// prototyp volání programu
dcl-pr cl_program extpgm('PTRCL');
    param like(parametr);
    soucet packed(10: 2);
end-pr;

for i = 1 to %elem(array);
    array(i) = i;
endfor;

callp cl_program (parametr: soucet); // vyvolání programu PRGCL
dump(a);
return;
```

Kompilujeme-li tento program volbou 14 v PDM (příkazem CRTBNDRPG), s předvolenými parametry, dostává program předvolenou aktivační skupinu – DFTACTGRP(*YES). Ve volném formátu není povolen příkaz CALL, proto se volání programu PTRCL provádí příkazem CALLP, který vyžaduje prototyp. V prototypu zadané vnitřní jméno volaného programu může být libovolné, je však nutné zadat skutečné jméno parametrem EXTPGM. V tomto případě jde o *dynamické volání programu*, přestože příkaz CALLP se zpravidla používá ke statickému volání procedur.