

HiSoft C 1.1

+GAMA

aneb perla z dob, kdy Spectrum bylo ještě mladé

Na úvod o Cčku

Něco o tomto programu už vyšlo v YS 11/99, proto se neztekejte, když se budu v něčem možná opakovat.

Existuje mezi lidmi představa, že spectrácké Cčko musí být nutně neplnohodnotné, že to není ono, že na PeCi je to mnohem lepší. Nemusí to být pravda. Abych vás přesvědčil, trochu ho popíšu. A začnu možná nelogicky právě jeho nedostatky.

Nedostatek číslo jedna - spectristé o Cčku nejevili valný zájem, takže se jeho původce HiSoft přestal starat o další vývoj a ačkoliv Cčko vzniklo už roku 1984, máme od něj pouhé dvě verze: 1.0 a 1.1, které se od sebe navíc liší jen v jakýchsi vychytených nedostatcích.

HiSofti se tak vůbec nedostali k tomu, dotáhnout některé plánované věci do konce, a tak se musíme spokojit s prozatímní situací: typy short, int a long jsou jedno a to samé 16ti bitové číslo, typy float a double nejsou implementovány. A to navzdory tomu, že hned do následující verze se sliboval short 8bitový, int 16bitový a long 32bitový a uvažovalo se i o implementaci plovoucí desetinné čárky. Ze zcela pochopitelných důvodů se proměnné, definované jako registry, zpracovávají jako automatické (prostě Z80 nemá tolik registrů a na funkci to stejně skoro nic nemění), a taková drobnost - místní proměnné se deklarují na začátku funkčního bloku a ne vnořených bloků. Toto poslední opatření má přispět k přehlednosti programu, ale většina dialektů Cčka to nedodržuje, při konverzi z jiných počítačů tedy budete asi muset ve zdrojáku hýbat s deklaracemi proměnných.

Druhá věc, kterou já osobně vidím jako nevýhodu, je příšerný editor, stejný jako editor HiSoft Pascalu, GENSu, inspirovaný řádkovými editory běžnými pod CP/M (podobný editor měl i EditAs, programátoři na tuhle hrůzu byli svého času zvyklí právě z CP/M).

Pochopitelně existuje možnost importu textu z jiného editoru nebo šance pro šikovné programátory nahradit stávající editor lepšími.

Někoho by možná děsilo, že Cčko na UNIXu pracuje se soubory sekvenčním způsobem a to Spectrácké to nebude umět (asi nechte Your Spectrum). Umí, původní verze pracovala s kazetou i microdrivem. Sekvenční soubor na micro-

drivu měl ovšem v hlavičce uvedenu délku 0 a soubor na kazetě byl rozsekán na 514 bytové sektory (2 bajty číslo sektoru, 512 bajtů data). Verze, kterou si upravili Rusové pro BETASHIT, používá soubory k nerozeznání od obyčejného bloku CODE. Vzhledem k tomu, že I/O rutina je jen jedna a je používána editorem, kompilátorem i přeloženým programem, nabízí se jednoduchá možnost, jak Cčko zpřístupnit i pro jiné systémy (D40/80).

Byli to rovněž Rusové, kdo se nespokojil s absencí reálných čísel a výsledkem byla krátká a jednoduchá knihovna, kterou tu taky někde najdete. Podobně existuje knihovna pro práci s pamětí Spectra 128 a jiné mnohé krásné. Dokonce i mnohé standardní funkce a konstanty dodávali sami HiSofti k Cčku jako knihovnu (taky tady někde je), například jednoduchou 32bitovou aritmetiku.

Nakonec zajímavost - C pro Spectrum bylo napsáno v Cčku (BDS C od L. Zolmana z BD Software). Po přeložení bylo disassemblováno a ručně v assembleru optimalizováno na krátkost i rychlost, stále ještě v něm zůstává několik původních funkcí BDS Cčka. Je to krásná ukáзка toho, že v Cčku je možné napsat skutečně všechno...

No a to by měl být konec plků, teď vás jistě zajímá, jak to Cčko vlastně vypadá z



hlediska programátora. Pokud si ale myslíte, že se přečtením tohoto článku naučíte v Cčku programovat, to jste si radši měli přečíst nějakou pohádku...

Něco o řádkovém editoru

V Cčku se obvykle píše malými písmeny, ENTER dává kódy 13, 10, jako konec řádky se bere kód 10, kód 13 se ignoruje (myslete na to při konverzi textu z jiných

editorů). Symbol shift s klávesou dává kód i těch symbolů, které normálně potřebují extend mód, například složené závorky. <=, <>, >= mají nové kódy 29, 31, 30 a dají se jako <=, != a >= použít normálně v programu. Symbol shift+I dává kód EOF (-1, 255, zobrazuje se jako chr\$ (137)). Protože kód 10 je konec řádku, dává caps shift+6 hodnotu 16.

Program se nemusí psát jen v editoru. Kratší věci můžete nařukat rovnou do kompilátoru, přeložit a spustit (trochu to připomíná okamžité provádění povelu v Basicu), je-li vám to málo, můžete se přepnout do módu přímého provádění a pak se rovnou provede každý napsaný řádek.

Z kompilátoru se do editoru vstupuje stiskem Edit a pak Enter. Při chybě při kompilaci se dá také stisknout Edit a chybná řádka se rovnou vyedituje.

Řádky jsou sice číslovány, ale jen pro interní potřebu editoru, očíslovávají se při čtení souboru z média. Nejlepší by bylo, kdyby někdo napsal nějaký šikovný full screen editor, kde by se nějakým číslováním a krkolomnými způsoby editace nemusel programátor vůbec zabývat (neříkal jsem už, že je ten editor fakt debilní?).

Povely pro editor se, jako jinde u HiSoftů, zadávají ve formátu C N1,N2,S1,S2 Enter, kde C je příkaz, N čísla a S stringy. Kromě povelu D a N nejsou parametry povinné.

I m,n - vkládání textu od řádku m,

krok n. Píšete a odentrováváte jednotlivé řádky, když vás to přestane bavit, stisknete Edit a Enter.

L m,n - list od řádku m do n (nebo všeho, bez parametrů).

K n - po kolika řádcích má listing čekat se scrollováním.

W m,n - listing textu na tiskárnu, lze ho breaknout.

V - zobrazí status: oddělovač, N1, N2, S1, S2, počáteční a koncovou adresu textu.

S,,d - nastaví oddělovač parametrů pro editor, nesmí to být mezera.

C - návrat do kompilátoru.

B - návrat do basicu, zpátky USR 25200.

D m,n - smaž řádky m až n.

M m,n - řádek m přesuň na novou pozici n.

N m,n - přečísluj řádky s počátkem m a krokem n.

P m,n,s - uloží text od řádku m do n (nebo celý, bez parametrů).

G,,s - na konec případného textu přihraje další, řádky čísluje s krokem 10.

Jméno se zadává ve formátu NAME pro kazetu a 1:NAME pro microdrive (1 je číslo drajvu). V BETASHITové verzi se postupuje podobně, výstup na kazetu je tam ale zcela zrušen. Čísla 1, 2, 3, 4 odpovídají mechanikám A, B, C, D. Při otevření existujícího souboru pro zápis se nejdřív starý obsah smaže (to v tomto Cčku platí při jakékoliv práci se souborem).

F m,n,f,s - v rozsahu m až n hledej řetězec f. Při nalezení přepne do editačního módu, kde krom jiného umožňuje řetězec nahradit řetězcem s, případně hledat další výskyt.

E n - editovat řádek n. Editace ale není normální editace - i při ní se používají povely:

šipka vpravo a vlevo - posun kurzoru vpravo a vlevo.

ENTER - konec editace, ponechání změn.

Q - konec editace, zrušení všech změn.
R - obnovení řádku tak, jak byl před editací.

L - výpis celého řádku (normálně část za kurzorem není vidět).

K - smazání znaku za kurzorem (normálně není vidět).

Z - smazání do konce řádku.

F - hledej další řetězec definovaný polem editoru F.

S - nahraď řetězec řetězcem, oba definované polem editoru F.

X - přepisování dalšího textu na konec řádku (přejde do módu Insert).

I - Insert, kurzor se mění na ,*‘, až do stisku Enter vkládá znaky.

C - kurzor se mění na ,+‘, až do stisku Enter přepisuje původní znaky novými.

Povely kompilátoru

Kromě nedostatků, které jsem zmínil v úvodu, neliší se HiSoft C 1.1 od definice a je tak kompatibilní s valnou většinou dialektů, nemá zabudovány žádné anachronismy, umí základní preprocesorové povely, podmíněnou kompilaci ovšem nezahrnuje jako takovou, místo toho má příkaz pro prohledávání knihovny.

Uvedu teď povely preprocesoru (můžou se vyskytovat v zdrojovém textu nebo je můžete zadávat rovnou do kompilátoru).

#define identifikátor makro

- definuje symbolická jména nebo konstanty, když ve zdrojáku najde váš nový identifikátor, chová se k němu jako k tomu, co jste zadali jako řetězec (je to takový token, umožňuje přejmenovávat funkce nebo měnit prostředí Cčka k obrazu svému). Například tyto řádky:

```
#define EOF -1
#define begin {
#define end ;}
```

#include

- příslušný řádek se nahradí textovým souborem. Bez parametru vkládá do kom-

pilátoru text z editoru. Do svého zdrojového textu můžete zahrnout i jiné soubory těmito povely:

```
#include filename
#include "filename"
#include <filename>
```

Navíc je implementováno tzv. prohledávání knihovny, je to náhrada podmíněné kompilace. Text se nekompiluje celý, ale z definované knihovny se berou jen ty funkce, na které se v předchozím textu odkazovalo. Proto se prohledávání knihovny umisťuje až na konec zdrojáku. **#include** může být jen v hlavním zdrojáku, ne v inkudovaném textu. Prohledávání se zapisuje takto:

```
#include ?filename?
```

#list

potlačit nebo zase povolit listing při kompilaci umožňují příkazy **#list+** a **#list-**.

#direct

při psaní textu přímo do kompilátoru se obvykle více řádků dá zkompilovat. Pokud chcete přímé vykonání každého samostatného řádku, zadejte povel **#direct+**, návrat k původnímu způsobu práce je **#direct-**. Direct mód umožňuje testovat části programu, měnit proměnné, vyvolávat funkce a spouštět smyčky...

#error

tento povel smaže chybová hlášení. Chyby se pak oznamují jen číslem bez slovního popisu, ale uvolní se tím paměť - někdy je tento povel vítanou záchranou.

#translate filename

- zadává se na začátku programu.

Překlad pak vyprodukuje spustitelný samostatný stroják, začínající na adrese 25200. Filename je jméno, pod kterým se má přeložený kód uložit.

Znak EOF se do kompilátoru vkládá stiskem SS+I, po jeho vložení se ukončí kompilace.

O Céčku

HiSoft C rozeznává 6 tříd symbolů: Identifikátory, klíčová slova, konstanty, řetězce, operátory a oddělovače. Poznámky se oddělují znaky ,/*‘ a ,*/‘, nesmí být vnořené, ale mohou obsahovat i konce řádků.

Identifikátor je posloupnost písmen a číslic, začínají písmenem, rozlišují se písmena malá a velká.

Klíčová slova se píší zásadně malými písmeny. Jsou to:

int, extern, else, char, register, for, float, typedef, do, double, static, while, struct, goto, switch, union, return, case, long, sizeof, default, short, break, continue, unsigned, auto, if.

Konstanty jsou integer, long, znakové a řetězce.

Integer konstanta dekadická je -32768

až 32767. Oktálová začíná nulou, hexadecimální začíná ,0x‘ nebo ,0X‘. Končí-li znakem ,!‘ nebo ,L‘, chápe se jako long. Znakové konstanty se píší do apostrofů, třeba ,X‘. Kromě toho existují tyto způsoby zápisu znaků:

\n - LF (nový řádek)

\t - tabulátor

\b - backspace

\r - CR (enter)

\f - FF (nová stránka)

\\ - zpětné lomítko

\‘ - apostrof

\DDD - DDD je oktálové číslo, udávající kód znaku.

Konstanty řetězcové se uzavírají do uvozovek, kompilátor je ukončuje nulou. Pozor, ,X‘ a ,X“ není totéž! Jedno je char, druhé string („pole charů“).

Konstanty s plovoucí čárkou by se chápaly jako double, ale nejsou implementovány.

Datové typy jsou char, int, float, double, celá čísla jsou ještě short, long a unsigned. HiSoft C short, int a long zpracovává všechny stejně, float a double nejsou implementovány.

Paměťové třídy jsou auto, static, extern, register a typedef..

Odvozené konstrukce jsou array (pole), funkce, pointer (ukazatel na adresu), struct (struktura), union (může obsahovat jakýkoliv z několika objektů).

Lvalue (L-hodnota) je výraz vztažený k objektu. Vysvětlím to na příkladu - máme ukazatel E na hodnotu 12345. *E je potom lvalue, třeba obsah paměti na adrese 12345. Typickou lvalue je třeba identifikátor víc napíšu u výrazů.

Jen krátce o konverzi typů: char na int - hodnota znaku. Pointer a int - lze sčítat i odčítat inty a pointery. Pointer - pointer=int. Int na unsigned. Konverze v aritmetice má jednoduchá pravidla - char nebo short se mění na int, je-li jeden operand unsigned, vše ostatní se mění taky na unsigned, nižší typ se převádí na vyšší, při přiřazení se hodnota vpravo konvertuje na typ hodnoty vlevo. Od nejvyšších k nižším jsou typy řazeny takto: Double, float, long, unsigned, int, short, char.

Výrazy

Unární operátory jsou tyto:

*výraz - pointer, výsledek je lvalue.

&lvalue - výsledek je pointer. Jde použít jen na proměnné a elementy pole.

-výraz - unární mínus.

!výraz - log. negace (1=nulový, 0=nenulový).

~výraz - jednotkový komplement.

++lvalue - objekt se inkrementuje před vyhodnocením.

lvalue++ - objekt se inkrementuje po vyhodnocení.

--lvalue - objekt se dekrementuje před vyhodnocením.

lvalue-- - objekt se dekrementuje po vy-

hodnocení.

cast(typ)výraz - konverze na daný typ. V jiných mutacích Cčka se slovo cast vynechává, proto při převodu programu z HiSoft C na tyto mutace musíte v úvodu uvést #define cast'.

sizeof výraz - vrací velikost v bytech.

sizeof(typ) - vrací velikost v bytech.

Multiplikační operátory jsou tyto: výraz*výraz - násobení.

výraz/výraz - dělení.

výraz%výraz - zbytek po dělení.

Aditivní operátory:

výraz+výraz - součet; pointer na objekt pole plus něco je pointer na jiný objekt pole.

výraz-výraz - rozdíl, viz sčítání, pointer-pointer=int (vzdálenost mezi poli).

Shift operátory:

výraz<<výraz - rotuje první výraz doleva o počet bitů daný druhým výrazem.

výraz>>výraz - rotuje první výraz doprava o počet bitů daný druhým výrazem.

Relační operátory:

výraz<výraz

výraz>výraz

výraz<=výraz

výraz>=výraz - všechny dávají 0 při FALSE, 1 při TRUE.

Srovnávací operátory:

výraz==výraz

výraz!=výraz - narozdíl od relačních mají nižší prioritu při vyhodnocování.

Bitové, logické a podmínkové operátory:

výraz&výraz - bitový and.

výraz&&výraz - logický and (1 jsou-li oba nenulové, jinak 0).

výraz~výraz - bitový xor.

výraz\výraz - bitový or.

výraz\\výraz - logický or.

výraz?výraz:výraz - je-li první výraz nenulový, je výsledkem druhý, jinak třetí výraz.

Přiřazovací operátory:

lvalue=výraz

lvalue+=výraz - je to jako lvalue=lvalue+výraz.

lvalue-=výraz

lvalue*=výraz

lvalue/=výraz

lvalue%=výraz

lvalue>>=výraz

lvalue<<=výraz

lvalue&=výraz

lvalue~=výraz

lvalue\\=výraz

Primární výrazy jsou:

identifikátor - pole dává pointer na první objekt, ne lvalue; funkce dává pointer na funkci.

konstanta

řetězec

(výraz) - typ a hodnota se bere z výrazu v závorkách, u lvalue jsou zbytečné.

výraz[index]

funkce (parametr,parametr,...)

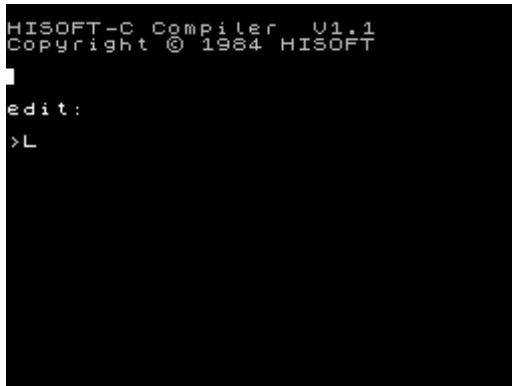
lvalue.identifikátor - lvalue je jméno struktury nebo unionu, výsledek je lvalue ukazující na jméno položky structu či unionu.

pointer->identifikátor - pointer je na struct nebo union, identifikátor jméno jeho člena.

Deklarace

Deklarace upřesňují interpretaci identifikátoru, dávají se před specifikátory typů. Zapisuje se jako:

specifikátor_třidy specifikátor_typu spe-



cifikátory_deklarace

Třídy jsou: auto, static, extern, register, typedef. Specifikátor třídy není povinný, potom se bere jako auto uvnitř funkce a jako extern vně funkce.

auto - proměnné lokální, při výstupu z bloku se uvolňují.

static - lokální, trvají stále a mohou zase do bloku vstupovat.

extern - trvají stále a mohou komunikovat mezi funkcemi. Pokud se povelem pro prohledání knihovny přilepují k programu funkce, které nevracejí int, musí být před voláním těchto funkcí (tedy na začátku programu) definován jejich typ, jinak budou přeloženy jako funkce vracející int.

register - programátor chce, aby pro tuto proměnnou byl vyčleněn přímo registr procesoru. Z80 ale jeho přání vytrvale ignoruje a proměnná se zpracovává jako automatická.

typedef - nevynezuje paměť pro proměnnou, ale definuje později použitelné identifikátory, které se stávají součástí deklarátoru.

Typy jsou: char, short, int, long, unsigned, float, double, typedef jméno, specifikátor struktury nebo unionu. Nezapomeňme, že short, int a long se chovají všechny jako int a float a double nejsou implementovány.

Seznamy deklarátorů se oddělují čárkou. Deklarátory se píší takhle:

identifikátor

(deklarátor)

*deklarátor

deklarátor()

deklarátor[výraz]

Například:

typ *D - pointer na typ

typ D() - funkce vracející typ; pozor, funkce nemohou vracet pole, struktury, uniony nebo jiné funkce, ale mohou

vracet pointery na ně; podobně neexistují pole funkcí, ale mohou být pole pointerů na funkce, třeba

```
int i,*ip,f(),*fip(),(*pfi());
```

deklaruje int i, pointer ip na int, funkci f vracející int, funkci fip vracející pointer na int a pointer pfi na funkci, která vrací celé číslo. S funkcí lze provádět jen její vyvolání nebo přiřazení její adresy, pokud není jméno funkce v pozici volání, generuje se pointer, třeba v programu

```
int f();
```

```
g(f)
```

se definice funkce g chápe jako

```
g(*fp());
```

```
int (*fp)();
```

```
{{*fp}();
```

, tedy f jako argument funkce g se chápe jako pointer na funkci f.

typ D[] - pole typů, je povoleno i víc specifikací vedle sebe, nejpraktičtější je to u vícerozměrného pole,

```
static int x[3][2][5];
```

deklaruje pole x jako statické pole intů velikosti 3x2x5 (třírozměrné pole dvourozměrných polí pětirozměrných polí intů), ale třeba také

```
int a[17],*ap[17];
```

deklaruje pole int čísel a pole pointerů na int čísla,

```
int *pa;
```

deklaruje pointer na int, pak

```
pa=&a[0];
```

nastaví pa na pointer elementu pole a[0], tedy pa obsahuje adresu a[0], navíc x=*pa

kopíruje obsah a[0] do x. Protože jméno pole je vlastně pointer na začátek pole (a nikoliv lvalue!), lze pa=&a[a] psát i jako pa=a, a[i] je vlastně *(a+i) a výrazy &a[i] a a+i jsou identické!

Struktury a uniony:

Struktura je posloupnost jmenovaných členů, z nich každý může být jakéhokoliv typu.

Union může obsahovat jakýkoli jeden nebo více z několika svých deklarovaných členů.

Struktury se nesmí jmenovat stejně jako jiné konstrukce (třeba stejně jako některý int). Struktury (i uniony) nesmí obsahovat sebe samu, ale mohou obsahovat pointer na sebe. Se strukturou se pracuje jen pomocí operátoru & a jejich členů, nemůže být přiřazována ani kopírována jako celek ani předávána funkcím či vracena funkcemi, funkcím lze předávat jen člen struktury nebo pointer na strukturu. Struktury jsou jen external nebo static, ne auto.

HiSoft C v současné verzi neumí bitová pole (fields) a vzhledem k tomu, že HiSoft již Spectrum opustil, nikdy je umět nebude. Lze to obejít bitovým posunem a bitovými operacemi.

Několik příkladů:

```
struct tnode {
```

```
char tword[20];
```

```
int count;
```

```
struct tnode *left;
struct tnode *right;
}s,*sp;
```

Struktura tnode obsahuje pole dvaceti charů, int a dva pointery na sebe sama, s je deklarace téže struktury (při opakovaní deklarace se nemusí znovu psát dlouhá část definice), *sp je pointer na strukturu tnode. Pak sp->count se vztahuje ke členu count struktury na kterou ukazuje sp, s.left se vztahuje k ukazateli na levý podstrom struktury s, s.right->tword[0] se vztahuje k prvnímu znaku členu tword pravého podstromu s.

Jiný příklad:

```
struct {
int x;
int *y;
}*p;
```

++p->x inkrementuje x a ne p, protože implicitně jsou priority brány jako ++(p->x); (++p)->x inkrementuje p před přístupem k x a (p++)->x inkrementuje p po přístupu k x.

Ještě něco jiného:

```
struct key{
char *hexword;
int keycount;
}keytab[nkeys];
```

přístup k členům pole je přes pointery, je to pole struktur typu key, každý prvek pole obsahuje pointer na char a int.

Se strukturou i unionem mohou být v podstatě prováděny jen dvě operace: jmenování jednoho z členů pomocí ‚.‘ nebo ‚->‘ a přiřazení adresy operátorem ‚&‘. U ‚.‘ a ‚->‘ je vpravo člen struktury a vlevo její jméno (před ‚.‘ lvalue, před ‚->‘ pointer nebo int).

Inicializace:

Deklarátor může definovat počáteční hodnotu, platí, že inicializátor nebo jejich seznam = výrazu nebo seznamu inicializátorů (to v případě inicializace pole nebo struktury, nevyjmenované členy jsou zaplácny nulami).

Externí definice:

Definují se jako všechno ostatní, přidává se adjektivum extern. Externí proměnné žijí po celou dobu trvání programu, definovat se dají i externí funkce. Zás příklad:

```
int max(a,c,b)
int a,b,c
{int m;
m=(a>b)?a:b;
return((m>c)?m:c);
}
```

int specifikuje typ, max(a,b,c) je deklarátor funkce, int a,b,c je seznam deklarací pro formální parametry, {...} je blok příkazů. Co tato funkce vrátí, je snad jasné - největší číslo ze tří do funkce vstupujících. Všimněte si, že v hlavičce standardní knihovny je funkce max napsána jinak - tam totiž není počet argumentů znám, jejich počet vstupuje jako další argument díky povelu ‚auto‘ mezi deklarátorem funkce ‚f(...)‘ a ‚int arg;{...}‘. Doporučuji nahlédnout do hlavičky standardní knihovny, jak přesně je to uděláno, vám do funkce přijde jako jediný parametr lva-

lue délka argumentů v bajtech a vy si zjistíte jak jejich počet (vydělíte si je délkou jednoho argumentu), tak pointer na ně (pomocí &lvalue).

Příkazy Cčka

Výrazový příkaz se píše jako výraz;

a slouží k přiřazení nebo volání funkce. Složený příkaz je sekvence příkazů (složených nebo výrazových) ve složených závorkách.

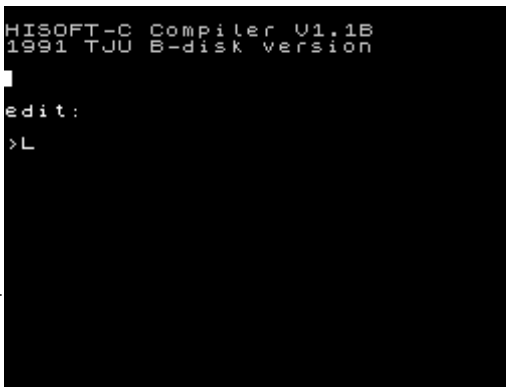
if (výraz) příkaz1;
if (výraz) příkaz1; else příkaz2;
je to podmíněný příkaz skoro stejný jako v basicu.

while (výraz) příkaz;
testuje před provedením příkazu, při nenulovém výrazu příkaz cyklicky provádí.

do příkaz while (výraz)
provádí cyklicky příkaz, po provedení testuje a opakuje, dokud je nenulový.

for (výraz1;výraz2;výraz3) příkaz;
výraz1 je inicializace, výraz2 je test, výraz3 je inkrementace. Kterýkoliv z nich může být vynechán. For je vlastně to samé jako

```
výraz1;
while (výraz2){
příkaz;
výraz3;
}
```



switch (výraz) příkaz;
příkaz je složený, prefixem ‚case výraz‘ se přiděluje příkazům hodnota, nebo se použije prefix ‚default:‘. Podle výsledku výrazu u switch se předá řízení buď té části příkazu, jejíž case odpovídá, nebo defaultní části příkazu. Po vykonání jedné části se porovnávají case u dalších částí příkazu, nechceme-li to, dá se ze switche vyskočit povelu break;

break;
ukončí cyklus while, do, for nebo switch.

continue;
ukončí jeden průchod smyčkou while,

do, for a jde na pokračovací část smyčky.

return; nebo return výraz;
vrátí se z funkce, výraz udává vrácenou hodnotu funkce.

goto identifikátor;
skočí na návěští dané identifikátorem. Návěští se zapisuje jako ‚identifikátor;‘.

;
je prázdný příkaz, hodí se, pokud chcete třeba prázdné tělo cyklu while nebo posunout návěští před závorku složeného příkazu.

inline (k1,k2,k3,...);
vkládá bajty strojového kódu, čísla 0-255 jako jeden byte, větší jako dvoubyte. Například:

```
#define CHAN_OPEN 0x1601
#define Id_a 0x3E
#define call 0xCD
inline (Id_a, 3, call, CHAN_OPEN, 0x22,
&c);
otevře kanál 3 a obsah registru HL uloží do proměnné c (0x22 je Id (NN),hl).
```

Standardní knihovna funkcí

Některé často používané přídatné funkce má HiSoft C zabudovány a není je tudíž vůbec třeba dotahovat z knihovny.

Ty ostatní jsou zahrnuty v knihovně stdio.h a stdio.l, přičemž stdio.h je jakási hlavička, která definuje některé konstanty a dvě funkce, u nichž není předem znám počet argumentů (max a min) a zařazuje se povelu ‚#include stdio.h‘ na začátek programu, zatímco stdio.l se zařazuje na konec programu pomocí příkazu pro prohledání knihovny ‚#include ?stdio.l?‘ a jsou z ní použity jen ty funkce, na které se programátor v předchozím textu odvolával.

Přestože je zabudována jen část funkcí, stojí určitě za to seznámit se se všemi.

int max(n,...) auto
vrací hodnotu největšího argumentu.

int min(n,...) auto
vrací hodnotu nejmenšího argumentu.

int abs(n)
vrací absolutní hodnotu.

int sign(n)
vrací -1, 0 a 1 pro čísla záporná, nulová a kladná.

char peek(adresa)
vrací obsah bytu na adrese adresa.

void poke(adresa,hodnota)
napoukaje nejnížší bajt hodnoty na

Programování

adresu `adresa`. Typ `void` obvykle znamená, že funkce nevrací žádnou hodnotu, Spectrum `void` sice neumí, ale tady to vyřeší jeden řádek `#define void int` a pak Ččko sežere i programy psané obvyklým způsobem, `void` je takto definován v `stdio.h`.

`int atoi(s)`
převede řetězec na číslo, mezery, tabulace a konce řádků ignoruje, umí znaménka.

`void qsort(list,num_items,size,cap_func)`
`char *list;`
`int num_items,size;`
`int (*cap_func)();`
Funkce se sice jmenuje Q-sort, ale to proto, že na UNIXu je quicksort součástí knihoven. Tohle je ve skutečnosti Shellův sort, ale kvůli možnosti používat zdrojáky z UNIXu bylo původní jméno zachováno.

`Položek` je `num_items`, mají velikost `size`, uspořádány jsou za sebou od pointeru `list`, seznam může být i dvourozměrné pole `char list [num_items][size]`. `Cap_func` je pointer na funkci, kterou se porovnají dvě položky v seznamu (třeba `strcmp` pro řetězce), bere dva pointerové argumenty, volá se `(*cap_func)(x,y)`; a vrátí `-1` když `*x<*y`, `0` když `*x==*y` a `1`, když `*x>*y`.

`char *strcat(base,add)`
`char *base,*add;`
fyzicky připojí kopii řetězce `add` na konec řetězce `base`, pozor, aby se něco nepřepsalo.

`int strcmp(s,t)`
`char *s,*t;`
porovná dva řetězce, při shodě vrátí nulu.

`char *strcpy(dest,source)`
`char *dest,*source;`
fyzicky zkopíruje řetězec `source` do `dest`, pozor, aby se něco nepřepsalo.

`unsigned strlen(s)`
`char *s;`
vrátí délku řetězce.

`int isalnum(c)`
`char c;`
vrátí `1`, jde-li o písmeno nebo číslo, jinak `0`.

`int isalpha(c)`
`char c;`
vrátí `1`, jde-li o písmeno.
Zabudovaná funkce.

`int isascii(c)`
`char c;`
vrátí `1`, jde-li o ASCII znak, přesněji řečeno jestli je menší než `128`...

`int iscntrl(c)`
`char c;`

vrátí `1` pokud je znak menší než mezera nebo roven `127` (copyright).

`int isdigit(c)`
`char c;`
vrátí `1` pro `,0'` až `,9'`.
Zabudovaná funkce.

`int isower(c)`
`char c;`
vrátí `1` pro malá písmena.
Zabudovaná funkce.

`int isprint(c)`
`char c;`
vrátí `1` u tisknutelných znaků (mezera až `126`).

`int ispunct(c)`
`char c;`
vrátí `1` u těch tisknutelných znaků, které nejsou ani číslo, ani písmeno.

`int isspace(c)`
`char c;`
vrátí `1` pro znaky mezera, nový řádek, tabulátor.
Zabudovaná funkce.

`int isupper(c)`
`char c;`
vrátí `1` pro velká písmena.
Zabudovaná funkce.

`char tolower(c)`
`char c;`
konvertuje velké písmeno na malé, jinak znak nemění.
Zabudovaná funkce.

`char toupper(c)`
`char c;`
konvertuje malé písmeno na velké, jinak znak nemění.
Zabudovaná funkce.

`char *calloc(n,size)`
`unsigned n,size;`
přídělí prostor pro `n` položek, každá velikosti `size`. Vrací pointer na začátek oblasti, jinak nulu (jako `,new'` v Packalu).

`void free(block)`
`char *block;`
zase blok uvolní. Předáváte kopii toho pointeru, který vám vrátila `calloc` (jako `,dispose'` v Packalu).

`char *sbrk(n)`
`unsigned n;`
vymezí `n` bajtovou oblast, kterou pak může používat funkce `calloc` pro heap. Na UNIXu systém sám zahýbe RAMěti tak, aby místo uvolnil, na Spectru je údaj o tom, kde místo uvolnit, přímo ve zdrojovém textu funkce. Ta standardní je napsána tak, že heap vytvoří jako static pole charů, tedy trvalou proměnnou uloženou mezi jinými proměnnými. Chcete-li heap jinde, přepište si laskavě sami.

`void swap(p,q,length)`
`char *p,*q;`
`unsigned length;`
prohodí dvě oblasti určené pointerem `p` a `q` o délce `length`, používá ji třeba `qsort`.
Zabudovaná funkce.

`void move(dest,source,length)`
`char *dest,*source;`
`unsigned length;`
něco jako inteligentní `ldir`, přesouvá oblast od pointeru `source` do pointeru `dest` o délce `length` a pracuje korektně i tehdy, když se obě oblasti překrývají.
Zabudovaná funkce.

`FILE *fopen(name,mode)`
`char *name,*mode;`
pozor, `name` i `mode` jsou pointerem na stringy, ne `char` (!), a to navzdory tomu, že `mode` je jednoznakový string! `Mode` může být `,r'` pro čtení a `,w'` pro zápis, `name` je jméno souboru, který se má otevřít, pokud otevíráte pro zápis již existující soubor, vymaže se jeho obsah.

Typ proměnné `FILE` je ve skutečnosti typedef `int` a je to ukazatel, který funkce vrátí pro potřebu jiných funkcí - potvrzuje typ souboru (zápis, čtení atd), při chybě je nulový.

Zabudovaná funkce.

`int fclose(fp)`
`FILE *fp;`
uzavře soubor označený pointerem `fp`. Po uzavření souboru se uvolní buffer, který systém pro vstup a výstup používal.
Zabudovaná funkce.

`int getc(fp)`
`FILE *fp;`
čte „znak“ ze souboru označeného pointerem `fp`. Ve skutečnosti nevrací `char`, ale `int`, protože potřebuje rozlišit kód `255` od `-1` (EOF, end of file).
Zabudovaná funkce.

`int ungetc(c,fp)`
`int c;`
`FILE *fp;`
vrátí znak `c` zpátky do souboru označeného pointerem `fp`. Jde to jen s jediným znakem, navíc funkce `ungetc` je použita funkcí `scanf`, takže po `scanf` už nejde `ungetc` použít (jde, ale musíte něco mezitím přečíst pomocí `getc`).
Zabudovaná funkce.

`int putc(c,fp)`
`int c;`
`FILE *fp;`
pošle znak `c` do souboru `fp`, znak taky vrátí jako svůj výsledek.
Zabudovaná funkce.

`int getchar()`
vrátí znak ze standardního vstupu „`stdin`“, na Spectru klávesnice. Vstup se děje přes buffer, text je možno opravovat

i použitím delete, vstup ukončuje enter, takže je to spíš takový basicovský input.

Zabudovaná funkce.

```
void exit(n)
uzavře všechny soubory a vyleze z programu. Parametr n vyvolá v nule „chybovou“ hlášku OK, jinak odpovídá číslu chybové hlášky, která se má zobrazit. Pro opuštění programu se dá použít i její „podfunkce“ _exit(n), ale vzhledem k tomu, že ta nechává bordel v souborovém systému, používejte jen exit(n), je to jistější.
```

```
char fgets(s,n,fp)
char *s;
int n;
FILE *fp;
čte řetězec s od souborového pointeru fp. Čtení ukončí buď když s dosáhne délky n, nebo když narazí na konec řádku.
```

```
void fputs(s,fp)
char *s;
FILE *fp;
nechá vystoupit řetězec od souborového pointeru fp.
```

```
char *gets(s)
char *s;
čte řetězec ze standardního „stdin“, tedy klávesnice.
```

```
void puts(s)
char *s;
nechá řetězec vystoupit na „stdout“, tedy obrazovku.
```

```
void printf(control,arg1,arg2,...)
char *control;
formátovaný tisk na „stdout“, tedy obrazovku, podle specifikace control. Řetězec control se normálně vytiskne tak, jak je, kromě konverzních údajů, které se uvozují znakem ‚%‘ a udávají, jak se mají vytisknout další argumenty. Znak ‚%‘ je možno normálně vytisknout zadáním ‚%%‘. A teď ty kon(tro)verzní údaje:
```

- %d - desítkové číslo
 - %o - oktálové číslo bez znaménka
 - %x - hexadecimální číslo bez znaménka
 - %n - desítkové číslo bez znaménka
 - %c - jeden znak
 - %s - řetězec ukončený znakem „\0“
 - %- - zarovnání vlevo (jinak se zarovnává doprava)
 - %0 - doplnit nulami místo mezer
 - %999 - (řetězec cifer) určuje minimální rozměr pole
 - %.999 - maximální počet znaků řetězce, které se vytisknou
 - %| - „dlouhá data“, tiskne bez omezení
- Zabudovaná funkce.

```
void fprintf(fp,control,arg1, arg2,...)
FILE *fp;
char *control;
jako printf, ale nikoliv na „stdout“, ale od souborového pointeru fp.
Zabudovaná funkce.
```

```
void sprintf(s,control,arg1,arg2,...)
char *s;
char *control;
jako printf, ale nikoliv na „stdout“, ale do řetězce s. Pozor, abyste si něco nepřepsali.
```

Zabudovaná funkce.

```
int scanf(control,arg1,arg2,...)
char *control;
(!) arg1,arg2... musí být pointery! Je to hodně divoká vstupní funkce, tak pozor: čte znak ze „stdin“, tedy klávesnice, konvertuje podle control a ukládá podle dalších argumentů do paměti.
```

Control obsahuje jednak prázdné znaky (mezera, tabelátor, CR, LF), odpovídající libovolnému počtu prázdných znaků na příslušné pozici (0-nekonečno), běžné znaky, které musí odpovídat následujícímu vstupnímu znaku.

Konverzní specifikace jsou tvořené znakem ‚%‘, případně ‚*‘, který potlačuje přiřazení, případně číslem a konverzním znakem. Konverzní specifikací se určuje, jak se bude interpretovat dané vstupní pole. Výsledek se ukládá do proměnné, na niž ukazuje pointer, argumenty tedy začínají operátorem &. Potlačující znak * způsobí přeskočení daného vstupního pole. Pole je ohraničeno buď prázdným znakem, nebo rozměrem pole(viz %999). Konverzní znaky jsou podobné jako u printf:

- d - očekává se desítkové číslo, pointer je na int.
 - o - očekává se oktálové číslo, pointer je na int.
 - x - očekává se hexadecimální číslo, pointer je na int.
 - h - obvykle značí short, v HiSoft C je pointer na int.
 - c - jeden znak, pointer je na char.
 - s - očekává se string, argumentem je pointer na řetězcové pole (musí být dost velké, aby se tam string vešel i s kódem „\0“).
- Funkce vrací počet úspěšně přiřazených položek nebo EOF.

```
int n;
char s[20];
scanf(„%d , %19s“,&n,s);
přečte celé číslo do n a řetězec (ne delší jak 19 znaků) do s, při vstupu musí být oba údaje odděleny čárkou. Údaj „%19“ sám ohlídá délku řetězce, argument s se nezadáva jako &s, protože pole je vlastně pointerem na svůj začátek! Už to sem píšu poněkolkráté...
```

Funkce scanf ve spectráckém Cčku neodpovídá přesně definici jazyka, je udělané spíš tak, aby odpovídalo této funkci tak, jak je dělána v Cčku pro UNIX.

```
int fscanf(fp,control,arg1,arg2,...)
char *control;
FILE *fp;
jako scanf, ale vstup se neděje ze „stdin“, ale ze souboru podle pointeru fp.
```

```
int sscanf(s,control,arg1,arg2,...)
char *s;
char control;
jako scanf, ale vstup se děje z řetězce s.
```

```
int rawin()
čte znak přímo z klávesnice bez vopíček jako jsou kurzory a echa na obrazovku.
Zabudovaná funkce.
```

```
int keybit()
Po stížení klávesy vrací 1 tak dlouho, dokud kód klávesy nepřetete pomocí rawin nebo neshodíte flag v systémových proměnných, oznamující stížení klávesy.
```

Zabudovaná funkce.

```
void long_multiply(c,a,b)
char *a,*b,*c;
násobí dvě 32bitová čísla, c=a*b. Čísla jsou uložena jako pole čtyř charů, nejméně významné bity jsou v poli [0], nejvýznamnější v poli[3] a jsou bez znaménka.
```

```
void long_add(c,a,b)
char *a,*b,*c;
sečte dvě 32bitová čísla, c=a+b.
```

```
void long_init(a,n1,n0)
char *a;
unsigned n1,n0;
vytvoří 32bitové číslo. Třeba long_init(a,0x1234,0x5678); vytvoří a=0x12345678.
```

```
void long_set(a,n,d)
char *a;
unsigned n,d;
vytvoří 32bitové číslo. Číslo n je 16 bitů a číslo d udává, kam se mají umístit. Třeba long_set(a,0x1234,1); vytvoří a=0x00123400.
```

```
void long_copy(c,a)
char *a,*c;
provede kopii 32bitového čísla, přiřazení c=a.
```

```
int rand()
vrací 16bitové náhodné číslo (int).
```

```
void srand(n)
n je int, funkce nastavuje seeds generátor pro náhodná čísla.
```

```
void plot(on,x,y)
int on,x,y;
bod na souřadnicích x a y zobrazí nebo smaže podle toho, je-li on nulové čili nic.
```

```
void line(on,dx,dy)
int on,dx,dy;
kreslí nebo maže čáru od poslední pozice do souřadnic dx a dy. Parametr on funguje stejně jako u plotu.
```

```
int ink(color)
```


- 32 bad type combination
- 33 bad operand type
- 34 need an lvalue
- 35 not a defined member of a structure
- 36 expecting a primary here
- 37 undefined variable
- 38 need a type name
- 39 need a constant expression
- 40 can only call functions
- 41 : does not follow a ? properly
- 42 Destination of an assignment must be an lvalue
- 43 need a : to follow a ? - check bracketing
- 44 need a pointer
- 45 illegal parameter type
- 46 RESTRICTION: Floating Point not implemented
- 47 cannot use this operator with float arguments
- 48 bad declaration
- 49 storage class not valid in this context
- 50 RESTRICTION: can't manage initializers yet
- 51 duplicate declaration of structure tag
- 52 use a predeclared structure for parameters
- 53 structure cannot contain itself
- 54 bad declarator
- 55 missing ')' in function declaration
- 56 bad formal parameter list
- 57 type should be function
- 58 generated code area is full
- 59 need an lvalue
- 60 LIMIT: no more memory
- 61 RESTRICTION: use assignment or move() to initialise automatics
- 62 Cannot initialise this (disallowed storage class)
- 63 Cannot initialise this (disallowed type)
- 64 too much initialisation data
- 65 wrong file, find another header

Reálná čísla

Jak už víte, HiSoft přestal po verzi 1.1 své C dále vyvíjet a tak zůstala spousta věcí, které měly být v dalších verzích doplněny, jen fíkcí. Už asi nikdy nebude doplněno zpracování typů float a double.

Snad vám ale pomůže tato knihovna pro plouvání desetinnou čárku, kterou si napsali Rucové. Není dlouhá, ale opisujte pečlivě...

```

/*****/
/*          */
/* REAL arithmetics */
/*          */
/* last changed */
/* 03.01.1991 */
/*          */
/*****/
#list-
typedef char real[5],*p_char,bool;
#define maxin 16
#define end_c 0x38
char s[maxin];
bool rezult;
toreal(rl,str)
real *rl;
char *str;
{ char sign;
  sign=*str;
  if(sign=='+' | |sign=='-')
  str++;
  wpoke(0x5b00,rl );
  wpoke(0x5b02,str);

```

```

inline(0x2a,0x5c5d,0xe5,
  0x2a,0x5b02,0x7e,
  0x22,0x5c5d,
  0xcd,0x2c9b,0xe1,
  0x22,0x5c5d);
if(sign=='-')
  inline(0xef,0x1b,end_c);
pop_a();
}div(a,b,c)
real *a,*b,*c;
{p_poke(a,b,c);
  push_bc();
  inline(0xef,5,end_c);
  pop_a();
}mult(a,b,c)
real *a,*b,*c;
{p_poke(a,b,c);
  push_bc();
  inline(0xef,4,end_c);
  pop_a();
}sub(a,b,c)
real *a,*b,*c;
{p_poke(a,b,c);
  push_bc();
  inline(0xef,3,end_c);
  pop_a();
}add(a,b,c)
real *a,*b,*c;
{p_poke(a,b,c);
  push_bc();
  inline(0xef,0xf,end_c);
  pop_a();
}bool equ(a,b) /* a=b */
real *a,*b;
{ wpoke(0x5b02,a);
  wpoke(0x5b04,b);
  push_bc();
  inline(6,0xe,0xef,0xe,end_c,
    0xcd,0x2dd5,
    0x32,&rezult);
  return rezult;
}bool lt(a,b) /* a<b */
real *a,*b;
{ wpoke(0x5b02,a);
  wpoke(0x5b04,b);
  push_bc();
  inline(6,0xd,0xef,0xd,end_c,
    0xcd,0x2dd5,
    0x32,&rezult);
  return rezult;
}bool ge(a,b) /* a>=b */
real *a,*b;
{ wpoke(0x5b02,a);
  wpoke(0x5b04,b);
  push_bc();
  inline(6,0xa,0xef,0xa,end_c,
    0xcd,0x2dd5,
    0x32,&rezult);
  return rezult;
}r_copy(a,b)
real *a,*b;
{ move(a,b,5);
}tostr(s,r)
char *s; real *r;
{wpoke(0x5b00,s);
  wpoke(0x5b02,r);
  push(0x5b02);
  inline(0xef,0x2e,end_c,
    0xcd,0x2bf1,0x2a,0x5b00,

```

```

  0xeb,0xed,0xb0,0xaf,
  0x12);
  tomin();
}scanr(r)
real *r;
{ scanf(,"\\n%s",s);
  toreal(r,s);
}tomin()
{inline(0xcd,0x16bf);
  p_poke(a,b,c)
  real *a,*b,*c;
  {wpoke(0x5b00,a);
    wpoke(0x5b02,b);
    wpoke(0x5b04,c);
  }push_bc()
  {push(0x5b02);
    push(0x5b04);
  }pop_a()
  {inline(0xcd,0x2bf1,
    0x2a,0x5b00,0x77,
    0x23,0x73,0x23,
    0x72,0x23,0x71,
    0x23,0x70);
  }push(x)
  unsigned x;
  {wpoke(0x5b10,x);
    inline(0x2a,0x5b10,0x5e,
      0x23,0x56,0xeb,
      0x7e,0x23,0x5e,
      0x23,0x56,0x23,
      0x4e,0x23,0x46,
      0xcd,0x2ab6);
  }wpoke(addr,data)
  unsigned addr,data;
  {poke(addr ,data%256);
    poke(addr+1,data/256);
  }poke(addr,val)
  {*cast(p_char)adr=val;}
  printr(r,lng)
  real *r; int lng;
  {int i; char es;
    tostr(s,r);
    if(lng)
    {es=1;
      for(i=0;i<lng;i++)
      {if(es es=s[i];
        fprintf(tek,"%c",es?s[i]:' ');
      }
    }
    else
      fprintf(tek,s);
  }#list+
  /*****/
  /*          */
  /* End Real Arithmetics */
  /*          */
  /*****/

```

Příloha: Standardní knihovna funkcí

Následuje „hlavička“ standardní I/O knihovny, obsahuje nejpoužívanější konstanty, definice typů pro knihovnu a funkce „min“ a „max“. Zařazuje se na ZÁČÁTEK programu, který knihovnu používá, повеlem #include „stdio.h“. Vlastní knihovna se pak dává na konec programu.

Programování

```

/*****
* Hisoft C */
/* Standard Function Library */
/* Header for ZX Spectrum */
*/
/* Copyright (C) 1984 Hisoft */
/* Last changed 10 Dec 1984 */
/*****
#list-
#define NULL 0 /* for use with pointers */
#define FALSE 0 /* for Boolean operations */
#define TRUE 1 /* " */
#define EOF -1 /* end of file value */
#define ERROR -1
#define void int /* to use for type of functions which re-
turn no value */
/* File system Structure */
typedef int FILE;
/* Storage Allocation Structure and Variables */
struct _header
{ struct _header * _ptr;
  unsigned _size;
};
typedef struct _header HEADER, * HEADER_PTR;
HEADER _base, * _allocp;
/* Function type forward declarations for non-int library
functions */
extern char *strcat(), *strcpy(), *calloc(),
  *fgets(), *gets(), *sbrk();
extern unsigned strlen();
/*two arithmetic functions which need to be declared
BEFORE they are used, because they are vari-
adic (take any number of arguments). This
means that if you include „stdio.h“ these func-
tions will always be compiled into your program.
You may wish to make a version of this file with-
out these two routines if you want to compile a lar-
ge amount of code and don't need them.*/ int
max(param_byte_count) auto
{ static int argc, *argv, max;
  argc = param_byte_count/2 - 1;
  argv = &param_byte_count + argc;
  max = -32767;
  while (argc--)
  {
    if (*argv > max) max = *argv;
    --argv;
  }
  return max;
}int min(param_byte_count) auto
{ static int argc, *argv, min;
  argc = param_byte_count/2 - 1;
  argv = &param_byte_count + argc;
  min = 32767;
  while (argc--)
  {
    if (*argv < min) min = *argv;
    --argv;
  }
  return min;
}#list+
/*****
* Hisoft C */
/* Standard Function Library */
/* End Header */
/*****

```

Následuje vlastní standardní I/O knihovna. Obsahuje základní 32bitovou aritmetiku, obsluhy vstupů a výstupů,

beepací a grafické procedury, vazbu na systém...

Umístuje se na KONEC programu, který knihovnu používá, a to nejlépe příkazem, který knihovnu prohledá a zařadí do kompilace jen ty její části, na které byl v hlavním programu odkaz (#include ?stdio.l?).

```

/*****
* Hisoft C */
/* Standard Function Library */
/* Version for ZX Spectrum */
*/
/* Copyright (C) 1984 Hisoft */
/* Last changed 20 Nov 1984 */
/*****
#list-
/* Some arithmetic functions */
/* min and max are in „stdio.h“ because they are varia-
dic */
int abs(n)
{ return n<0 ? -n : n ;
}int sign(n)
{ return n ?
  ( n<0 ? -1 : 1 ) : 0 ;
}/* An illustration of how to grub around in the store */
typedef char * __char_ptr;
int peek(address)
{ return * cast(__char_ptr) address;
}void poke(address, value)
{ * cast(__char_ptr) address = value;
}/* Format conversion routine - ASCII to binary integer
*/
int atoi(s)
char *s;
{ static int c, value, sign;
  while (isspace(*s)) ++s;
  value = 0;
  sign = 1;
  if (*s == '-') { ++s; sign = -1; }
  else if (*s == '+') ++s;
  while (isdigit(c = *s++)) value = 10 * value + c - '0';
  return sign * value;
}/* Sorting function - a Shell sort */
void qsort(list, num_items, size, cmp_func)
char *list;
int num_items, size;
int (*cmp_func)();
{ static unsigned gap, byte_gap, i;
  static char *p;
  for (gap = num_items >> 1; gap > 0; gap >>= 1)
  {
    byte_gap = gap * size;
    for (i = gap; i < num_items; ++i)
      for (p = list + i * size - byte_gap; p >= list; p -= by-
te_gap)
      {
        if ((*cmp_func)(p, p + byte_gap) <= 0) break;
        swap(p, p + byte_gap, size);
      }
  }
}/* String Handling Functions */
char *strcat(base, add)
char *base, *add;
{ static char *result;
  result = base;
  while (*base) ++base;
  while (*base++ = *add++);
  return result;
}

```

```

}int strcmp(s, t)
char *s, *t;
{ while (*s == *t)
  {
    if (! *s) return 0;
    ++s; ++t;
  }
  return *s - *t;
}char *strcpy(dest, source)
char *dest, *source;
{ static char *result;
  result = dest;
  while (*dest++ = *source++);
  return result;
}unsigned strlen(s)
char *s;
{ static unsigned length;
  length = 0;
  while (*s++) ++length;
  return length;
}/* Character Test and Manipulate Functions */
/* NB - the common ones are built-in for efficiency */
int ispunct(c)
char c;
{ return isprint(c) & ! isalnum(c);
}int isalnum(c)
char c;
{ return isalpha(c) | isdigit(c);
}int isascii(c)
char c;
{ return c < 0x80 ;
}int iscntrl(c)
char c;
{ return c < , , | c == ,\177' ;
}int isprint(c)
char c;
{ return c >= , , & c < ,\177' ;
}/****** FILE SYSTEM *****/
char *fgets(s, n, fp)
char *s;
int n;
FILE *fp;
{ static int c;
  static char *cs;
  cs = s;
  while (-n > 0 && (c = getc(fp)) != EOF)
    if ((*cs++ = c) == ,\n') break;
  *cs = ,\0';
  return ((c == EOF && cs == s) ? NULL : s);
}char *gets(s)
char *s;
{ static int c;
  static char *cs;
  cs = s;
  while ((c = getchar())
    != EOF && c != ,\n')
    *cs++ = c;
  *cs = ,\0';
  return
    ((c == -1 && cs == s) ?
    NULL : s);
}void fputs(s, fp)
char *s;
FILE *fp;
{ static int c;
  while (c = *s++) putc(c, fp);
}void puts(s)
char *s;
{ while (putc(*s++));
}/* Storage Allocation and Freeing (Heap Management)

```

```

    */
char *calloc(n, size)
    unsigned n, size;
{ static HEADER *p, *q;
  static unsigned nbytes;
  nbytes = (n * size + (sizeof(HEADER) - 1)) / size-
    of(HEADER) + 1;
  if ((q = _allocp) == NULL) /* no free list */
  {
    _base._ptr = _allocp = q = &_amp_base;
    _base._size = 0;
  }
  p = q->_ptr;
  while (TRUE)
  {
    if (p->_size >= nbytes) /* big enough */
    {
      if (p->_size == nbytes) q->_ptr = p->_ptr; /*
        just right size */
      else
      {
        /* split block and allocate tail
        */
        p->_size -= nbytes;
        p += p->_size;
        p->_size = nbytes;
      }
      _allocp = q;
      return cast(__char_ptr) (p+1);
    }
    if (p == _allocp) /* wrapped around free list */
    {
      if ((p = cast(HEADER_PTR) sbrk(nbytes * size-
        of(HEADER))) == ERROR)
        return NULL;
      p->_size = nbytes;
      free(p+1);
      p = _allocp;
    }
    q = p;
    p = p->_ptr;
  } /* end while TRUE */
}void free(block)
char *block;
{ static HEADER *p, *q;
  p = cast(HEADER_PTR) (block - 1);
  for (q = _allocp; !(p > q && p < q->_ptr); q = q->_ptr)
    if (q >= q->_ptr && (p > q || p < q->_ptr)) break;
  if (p + p->_size == q->_ptr)
  {
    p->_size += q->_ptr->_size;
    p->_ptr = q->_ptr->_ptr;
  }
  else p->_ptr = q->_ptr;
  if (q + q->_size == p)
  {
    q->_size += p->_size;
    q->_ptr = p->_ptr;
  }
  else q->_ptr = p;
  _allocp = q;
}#define HEAPSIZ 1000
char *sbrk(n)
    unsigned n;
{ static char p,
  heap[HEAPSIZ],
  heap_ptr=heap;
  if (heap_ptr+n > heap+HEAPSIZ) return ERROR;
  p=heap_ptr;
  heap_ptr += n;
  return p;

```

```

}/* Pseudo-Random Number Generator */
/* Adapted from „Learning to Program in C“ by Thomas
  Plum. */
char _rnum[4];
void srand(n)
{ long_init(_rnum, 0,n);
}int rand()
{ static char k[4];
  long_init(k, 0x41c6,0x4e6d);
  long_multiply(_rnum, _rnum, k);
  long_init(k, 0,0x3039);
  long_add(_rnum, _rnum, k);
  return (_rnum[1] << 8) + _rnum[0];
}/* Some Functions for 32 bit integer arithmetic */
void long_multiply(c, a, b)
  char *a, *b, *c;
{ static char x[4], product[4];
  int i, j;
  long_set(product, 0,0);
  for (i = 0; i < 4; ++i)
    for (j = i; j >= 0; -j)
    {
      long_set(x, a[i-j] * b[j], i);
      long_add(product, product, x);
    }
  long_copy(c, product);
}void long_add(c, a, b)
  char *a, *b, *c;
{ unsigned u, i;
  u = 0;
  for (i = 0; i < 4; ++i)
  {
    u += *a++ + *b++;
    *c++ = u & 0xff;
    u >>= 8;
  }
}void long_init(a, n1, n0)
  char *a;
  unsigned n1,n0;
{ a[0] = n0 & 0xff;
  a[1] = n0 >> 8;
  a[2] = n1 & 0xff;
  a[3] = n1 >> 8;
}void long_set(a, n, d)
  char *a;
  unsigned n, d;{
  static int i;
  for (i=0; i<4; ++i) a[i] = 0;
  a[d] = n & 0xff;
  if (d < 3) a[d+1] = n >> 8;
}void long_copy(c, a)
  char *a, *c;
{ move(c, a, 4);
}/** System Interface ***/
exit(n)
{ inline(0xcd,25236);
  _exit(n);
}_exit(n)
{ inline(0xe1,0xe1,0xe1,
  0x2b,0xc3,0x55,0);
}/* Spectrum Graphics and
  Sound Functions */
plot(on,x,y)
{ _setover(on);
  inline(0xdd,0x46,4,
    0xdd,0x4e,6,
    0xcd,0x22e5);
}line(on,dx,dy)
{ static sx,sy,de,bc;
  _setover(on);

```

```

sx=sy=1;
if (dx<0)
{
  dx= -dx;
  sx= -1;
}
if (dy<0)
{
  dy= -dy;
  sy= -1;
}
de= (sy<<8)+sx;
bc=(dy<<8)+dx;
inline(0xed,0x5b,&de,
  0xed,0x4b,&bc,
  0xcd,0x24ba);
}beep(duration,pitch)
  unsigned duration,pitch;
{ static ft;
  if (! pitch)
    for(ft=0;ft<duration;++ft)
      for(pitch=4630;++pitch);
  else {
    ft=duration*pitch/10;
    pitch=pitch/10;
    _beep(ft,cast(unsigned)43750/pitch-30);
  }
}paper(i)
{ return _colour(17,i);
}ink(i)
{ return _colour(16,i);
}cls()
{ inline(0xcd,0xd6b);
  _setover(on)
  { printf(„025%c“,on?0:1);
  }_beep(DE,HL)
  { static de,hl;
    de=DE;
    hl=HL;
    inline(0xdd,0xe5,
      0xed,0x5b,&de,
      0x2a,&hl,
      0xcd,0x3b5,
      0xdd,0xe1);
  }_colour(h,i)
  { if (i<0 | i>7) return -1;
    putc(h,2);
    putc(i,2);
    return i;
  }#list+
  /******
  /* Hisoft C */
  /* Standard Function Library */
  /* End */
  /******

```

Název: HiSoft C 1.1
Typ: programovací jazyk
Producent: Hi Soft
Hodnocení:

75 %