Jazyk C

Program v jazyku C má následující strukturu:
Direktivy procesoru
Globální definice (platné a známé v celém programu)
Funkce
    Hlavička funkce
    Tělo funkce – je uzavřeno mezi složené závorky {}
        Lokální definice (platné a známé jen v této funkci)
        Příkazy
…… funkcí může být více za sebou – nesmějí ale být vnořovány do sebe
hlavní funkce – je v programu právě jedna. Provádění programu při jeho spuštění začne prvním příkazem v těle hlavní funkce.

Některé části mohou chybět, pokud je nepotřebujeme. Vždy však musí existovat hlavní funkce, vždy také potřebujeme připojit k našemu programu některé knihovny direktivou include.
Zdrojový kód programu musíme přeložit kompilátorem. Překlad se skládá ze dvou fází, v první procesor plní direktivy (pozná je podle toho, že začínají znakem #, na konci není středník). Pak proběhne vlastní překlad. Pokud je úspěšný, lze program spustit.

Údaje, s nimiž program pracuje, jsou buď **konstanty nebo proměnné**. Proměnné musí být v programu definovány, což znamená, že **musí** být uvedeno jejich jméno a datový typ. Teprve pak se mohou jejich jména objevit v příkazech a výrazech. Jména (identifikátory) volíme zpravidla tak, aby vypovídala o obsahu proměnné. Používáme malá písmena , číslice a podtržítko (číslice nesmí být první); upřednostňujeme české názvy. To nám usnadní rozlišování mezi jmény (identifikátory) a klíčovými slovy, která jsou anglicky a slouží ke specifikaci příkazů nebo typů. Jazyk C rozlišuje velká a malá písmena, takže Prog, prog a PROG jsou různá jména. Pro oddělování klíčových slov, identifikátorů a konstant slouží oddělovače (mezera nebo přechod na nový řádek). Všude, kde mohou být oddělovače, může být komentář. Komentář začíná znaky /* a končí */ .
/* toto je komentar */
Jednoduché datové typy:
int pro celočíselné proměnné
float pro racionální proměnné
U konstant je typ dán způsobem zápisu: 4 je typu int, 4.1 je typu float. Existují i jiné typy, ale to necháme na později.
Deklarace celočíselných proměnných, kterým jsme dali jména a,b,c, budou tedy vypadat takto:
int a,b,c;
float d; je deklarace proměnné d, která může obsahovat racionální číslo (s desetinnými místy)
Strukturované datové typy: zatím známe jen pole, popíšeme později

Výrazy jsou tvořeny operandy (těmi může být proměnná, konstanta nebo jiný výraz) a **operátory**
Operátorů je v jazyku C velmi mnoho, nejdůležitější je operátor **přiřazení** =
Zápis a=b; znamená : obsah proměnné b přiřaď proměnné a
Operátory **aritmetické** jsou +, - ,*, / a %. Význam + a – je zřejmý, * je pro násobení, / je operátor dělení, jehož význam se ale liší v závislosti na operandech: buď jde o celočíselné dělení , nebo obvyklé dělení v případě alespoň jednoho neceločíselného operandu.

Operátor % (čteme modulo) znamená zbytek po dělení. Zvláštními operátory jsou unární (mají jen 1 operand) operátory ++ (přičtení jedničky) a -- (odečtení jedničky, které mohou navíc být psány dvojím způsobem.
a++ je postfixový zápis, ++a prefixový
Příklad:

```
a=10;
x=++a; /* x bude mít hodnotu 11, proměnná a taky */
y=a--;  /* y má hodnotu 11, a hodnotu 10 */
```

Operátory **relační** jsou: <, >, <=, >=, == (rovnost), != (nerovnost)
Operátory **logické**:
|| logický součet
&& logický součin
! negace

Příkazy.
Napíšeme-li za výraz středník, stane se z něj příkaz.

```
x=0;
a++;
```

Příkazy můžeme sdružovat do bloků (složených příkazů). Začátek a konec bloku je vymezen složenými závorkami. Složené příkazy používáme tam, kde smí být použit pouze jeden příkaz, ale potřebujeme jich více. Za uzavírací složenou závorku nepíšeme středník.

Řízení toku programu

Příkaz if

má dvě podoby:

```
if (výraz) příkaz    nebo
if (výraz) příkaz1
else příkaz2
```

Složitější rozhodovací postup můžeme realizovat konstrukcí if else if. Každé else se váže vždy k nejbližšímu předchozímu if.

Příkaz while.

```
while(výraz)
        příkaz
```

Výraz za while představuje podmínku pro opakování příkazu. Není-li podmínka splněna už na začátku, nemusí se příkaz provést ani jednou. Je-li splněna, příkaz se provede a po jeho provedení se znovu testuje podmínka pro opakování cyklu.

Příkaz do-while

Zajistí aspoň jedno provedení těla cyklu, protože podmínka opakování se testuje na konci cyklu.

```
do
        příkaz
while(výraz)
```

Příkaz for

```
for(inicializační_výraz;podmíněný výraz;opakovaný výraz)
        příkaz
```

je ekvivalentní zápisu:
Inicializační výraz;
while(podmíněný výraz)
{
	příkaz
	opakovaný výraz
}

Inicializační výraz může být vypuštěn, zůstane po něm však středník. Stejně může být vynechán i podmíněný výraz a opakovaný výraz.
Příkaz continue je možno použít ve spolupráci se všemi uvedenými typy cyklů. Ukončí právě prováděný průchod cyklem a pokračuje novým průchodem.
Příkaz **break** je rovněž možno použít se všemi typy cyklů. Znamená opuštění cyklu; program pokračuje příkazem za koncem cyklu.

Příkaz goto a návěští
Příkaz goto přenese běh programu na místo označené návěštím (identifikátor ukončený dvojtečkou). Jsou situace, kdy může být výhodný, např. chceme-li vyskočit z vnitřního cyklu z více vnořených cyklů.

Prázdný příkaz
;
Použití všude tam, kde je prázdné tělo.

Funkce

Každá funkce musí mít **definici** a
- má určeno jméno, pomocí kterého se volá
- může mít parametry, v nichž předáme data, na kterých se budou vykonávat operace
- může mít návratovou hodnotu poskytující výsledek
- má tělo složené z příkazů, které po svém vyvolání vykoná. Příkazy jsou uzavřeny ve složených závorkách {}

Příklad:
```
int max(int a, int b)		/* hlavicka */
{				 /* telo funkce */
	if (a>b)
		return a;
	return b;
}
```

Nevrací-li funkce žádnou hodnotu, píšeme void. Nepředáváme-li data, uvádíme jen kulaté závorky nebo void.
Neznáme-li definici funkce a přesto ji chceme použít, musíme mít **deklaraci funkce (prototyp),** která určuje jméno funkce, paměťovou třídu a typy jejích parametrů. Na rozdíl od definice funkce již neobsahuje tělo a je vždy ukončena středníkem.
int max(int a, int b); nebo jen
int max(int,int);

Volání funkcí
Obecně: jmeno_funkce(seznam skutečných parametrů);
Nemá-li žádné parametry, musíme napsat (). Parametry se vždy předávají hodnotou, jsou tedy následně překopírovány do formálních parametrů funkce. Počet skutečných parametrů by měl být shodný s počtem formálních parametrů a musejí být stejného typu.
Jazyk C dovoluje používání funkcí rekurzivních (funkce ve svém těle obsahuje volání sama sebe).

Pole
Pole je strukturovaný datový typ, tvořený složkami stejného datového typu.
Na složky se odvoláváme jménem pole a indexem (pořadím) složky v poli. Složky jsou v poli vždy indexovány počínaje nulou. Pole se definuje typem složek, názvem pole a počtem složek uvedeným v hranatých závorkách. Tedy na příklad
int a[10]; /* definuje pole se jménem a o deseti složkách */
Počet složek se vždy uvádí konstantou. Na složku s indexem např. 3 se odvoláme a[3]. Zde může v hranatých závorkách být i proměnná, dokonce i výraz, po jehož vyhodnocení ale musíme dostat celé číslo, které leží v rozmezí 0 až počet složek – 1. Nekontroluje se, že index nepřesahuje hranice pole, je za to odpovědný programátor. S polem jako celkem nelze provádět žádné operace, většinou používáme při práci cykl for. Na příklad, chceme-li, aby všechny složky pole a měly hodnotu nula, napíšeme:
n=10;
for (i=0;i<n;i++) a[i]=0;
Mají-li složky pole na začátku mít určité hodnoty, můžeme to zařídit už v definici takto:
int a[5]={1,2,3,4,5}; Pokud je hodnot ve složených závorkách více, než odpovídá rozměru pole, ohlásí překladač chybu. Je-li jich méně, doplní nuly.
Pole jako parametr funkce: Samotné jméno pole představuje adresu pole, takže zde jde vlastně o předání odkazem. Vše je zřejmé z příkladu:

```
void bubble(int p[],int n)   /* hlavička funkce */
{ … telo funkce }
int main()
{
int pole[20];
bubble(pole,pocet);          /* volání funkce */
}
```

Vstupy a výstupy
Vstupní a výstupní operace nejsou součástí jazyka C. Používáme proto knihovní funkce. Předpokladem je direktiva #include <studio.h>, protože knihovna stdio.h obsahuje funkci scanf pro formátovaný vstup a printf pro formátovaný výstup. Standardní vstup je z klávesnice, výstup na obrazovku.
Obě funkce mají proměnný počet parametrů. Ten je určen prvním parametrem- formátovacím řetězcem. Formátovací řetězec funkce printf může obsahovat dva typy informací. Jednak jde o běžné znaky, které budou vytištěny, dále pak speciální formátovací sekvence znaků začínající % . K tisknutelným znakům patří také escape sekvence, např. \n pro nový řádek. scanf se liší tím, že formátovací řetězec smí obsahovat jen formátovací sekvence, a tím, že druhým a dalším parametrem je vždy adresa proměnné.

Formátovací sekvence (printf): povinně se uvádí příznak typu (d nebo i pro int, f pro float). Dále můžeme zadat ještě spoustu jiných věcí, spokojíme se s tím, že lze zadat šířku, třeba %5d znamená, že celé číslo má být zobrazeno na 5 míst,

%6.2f znamená, že racionální číslo zobrazíme na 6 míst, z toho 2 desetinná
Formátovací sekvence (scanf): typ je rovněž d nebo i pro integer, f pro float. A to nám stačí.

U standardního výstupu se musíme ještě postarat o to, abychom si ho stihli přečíst. Lze toho dosáhnout různě, jedna z možností je funkce getch(), kde neuvádíme parametr. Pak program čeká, až stiskneme libovolnou klávesu. Předpokladem je #include <conio.h>.

Preprocesor

Preprocesor zpracuje zdrojový text programu před překladačem, vypustí komentáře, provede záměnu textů, např.identifikátorů konstant za odpovídající číselné hodnoty a vloží texty ze specifikovaných souborů. Příkazy pro preprocesor začínají znakem # a nejsou ukončeny středníkem. Nejdůležitějšími příkazy jsou #define a #include. #define ID hodnota
říká, že preprocesor nahradí všude v textu identifikátor ID specifikovanou hodnotou, např.
#define PI 3.14159
#include <stdio.h>
znamená příkaz vložit do zdrojového textu funkce vstupu a výstupu ze systémového adresáře.
#include "filename"
Preprocesor vloží text ze specifikovaného souboru v adresáři uživatele.
Některé standardní knihovny:

stdio.h	funkce pro vstup a výstup
stdlib.h	obecně užitečné funkce
string.h	práce s řetězci
math.h	matematické funkce
conio.h	funkce getch