

1. Minimální program

– do souboru main.c uložíme následující kód a pomocí „F9“ ho zkompilujeme a spustíme:

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

```
int main()
{
    printf("Hello world!\n");
    return 0;
}
```

Vypíše na obrazovku „Hello world!“ bez uvozovek a odřádkuje.

Ukončí program. Nula znamená, že vše proběhlo bez chyby.

2. Výpis do konzole

```
printf("Text");
```

– vypíše na obrazovku slovo Text (bez uvozovek)

– pokud chceme vypsát některý speciální znak nebo znak se speciálním významem, musíme ho tzv. oescapovat tak, že před něj napíšeme zpětné lomítko (\):

\n	Nový řádek (new line)
\t	Tabulátor (několik mezer za sebou)
\a	Pípnutí (alert)
\"	Dvojitá uvozovka
\\	Zpětné lomítko

```
printf("Text %d text %d ...", číslo1, číslo2, ...);
```

– vypíše text v uvozovkách, ale znaky „%d“ nahradí postupně číslem 1, 2 atd.

– znak za procentem určuje, v jakém formátu se má číslo vypsát:

%d	Vypíše číslo v desítkové soustavě.
%o	Vypíše číslo v osmičkové soustavě.
%x	Vypíše číslo v šestnáctkové soustavě malými písmeny.
%X	Vypíše číslo v šestnáctkové soustavě velkými písmeny.
%%	Vypíše znak procenta.

3. Komentáře

= texty ignorované překladačem

– používají se pro psaní poznámek nebo zakomentování části kódu, který nechceme vykonat

Řádkový komentář

– začíná dvěma lomítky a končí koncem řádku

– lze ho napsat na prázdný řádek i za příkaz

```
// komentář na prázdném řádku
```

```
printf("Hello world!"); // komentář za příkazem
```

Blokový komentář

- začíná znaky „/*“ a končí znaky „*/“
- mezi hvězdičkami může být libovolný počet řádků
- může obsahovat i řádkové komentáře

```
/*Komentář*/  
/*  
    Několika řádkový  
    Komentář obsahující  
    // řádkový komentář  
*/
```

2. lekce

4. Celočíselné proměnné

Definování proměnné

```
int i, cislo1;  
int j;
```

- po definování proměnné je v ní náhodná hodnota, před prvním použitím např. ve výpočtu nebo výpisu je potřeba do ní uložit nějakou hodnotu

Uložení hodnoty do proměnné

```
i = 5; // v proměnné i je 5  
cislo1 = i * 2 - 6; // v proměnné cislo1 je 5 (5 * 2 - 6)
```

Načtení hodnoty do proměnné z klávesnice

```
scanf("%d", &i);  
scanf("%d %d", &i, &cislo);
```

- před názvy proměnných se musí psát ampersand (&) !!!
- jednotlivé proměnné ve formátovacím řetězci oddělovat vždy jen mezerou, ačkoliv je možné používat i jiné znaky, ale pokud je pak uživatel přesně nezadá, tak se program začne chovat neočekávaně
- při čtení znaků z klávesnice se jako mezera interpretuje jakýkoliv **bílý znak** (mezera, konec řádku, tabulátor)
- znaky se nečtou přímo z klávesnice, ale z **klávesnicového bufferu**, takže pokud jsme při předchozím psaní na klávesnici toho napsali více, tak se text navíc použije pro následující volání funkce scanf

Adresa proměnné

- proměnné se ukládají do operační paměti (RAM)
- při programování se na proměnné odkazujeme jejich názvy, ale počítač používá jejich adresy
- aktuálně použitou adresu proměnné zjistíme přidáním ampersandu před název proměnné, např. &i
- adresa proměnné může být při každém spuštění programu jiná
- adresa proměnné je pořadové číslo prvního bytu proměnné v paměti
- byte (česky bajt) = 8 bitů; značení 1 B = 8 b
- do jednoho bytu se vejde pouze číslo od 0 do 255
- např. proměnná int má 4 byty, takže se do ní vejde číslo od 0 do 4 294 967 295 (cca 4.3 miliardy)

```
int i;  
i = 5;  
printf("Hodnota proměnné i: %d\n", i); // vypíše 5  
printf("Adresa proměnné i: %d\n", &i); // náhodné číslo, např. 23624
```

5. Celočíslné dělení

- celočíselné dělení je dělení ve stylu „13 děleno 5 je 2 zbytek 3“
- celočíselné dělení se dělá pomocí znaku lomítka (/), zbytek po dělení se nazývá **dělení modulo** a značí se procentem (%)

```
i = 13 / 5; // do proměnné i se uloží 2
i = 13 % 5; // do proměnné i se uloží 3; čteme 13 děleno modulo 5
```

6. Desetinná čísla

- na počítači se zapisují vždy s desetinnou tečkou, místo české čárky (např. 3.52 místo 3,52)
- definují se pomocí klíčového slova **float** (používá se ve většině programovacích jazyků)
- desetinná čísla se na počítači ukládají v různých formátech; slovo float znamená, že se číslo uloží tzv. s plovoucí (float) desetinnou tečkou
- v printf a scanf se místo %d použije %f

```
float cislo; // definování proměnné cislo
cislo = 3.52; // uložení čísla 3,52 do proměnné cislo
printf("%f", cislo); // vypsaní čísla
scanf("%f", &cislo); // načtení desetinného čísla z klávesnice
```

- pokud chceme určit, kolik má funkce printf vypsat číslic, používáme modifikovanou syntaxi %f:

```
printf("%7.3f", cislo);
```

- vypíše vždy alespoň sedm znaků (započítává se i desetinná tečka)

- zokrouhlí číslo tak, aby mělo tři desetinná místa

- pokud je číslo příliš krátké, vypíšou se před něj mezery tak, aby mělo sedm znaků; pokud je naopak číslo delší než sedm znaků, tak se vypíše delší, ale pořad bude mít tři desetinná místa

Například

```
printf("%6.2f\n", 123.45);
printf("%6.2f\n", 1.5);
printf("%6.2f\n", 2.47612);
printf("%6.2f\n", 17);
printf("%6.2f\n", 111222.6666);
```

vypíše

```
123.45      ... číslo má přesně 6 znaků
  1.50      ... číslu se doplnila nula a přidaly se před něj dvě mezery
  2.48      ... číslo se zaokrouhlilo
 17.00      ... číslu se doplnila desetinná místa
111222.67   ... číslo bylo příliš dlouhé, tak se vypsalo celé (říkáme, že přeteklo)
```

- pokud bychom chtěli před kratší čísla vypsat místo mezer nuly, tak se místo "%6.2f" zadá "%06.2f"

```
printf("%06.2", 1.37); // vypíše 001.37
```

- podobně lze tyto zápisy použít i pro celá čísla, jen se vynechá počet desetinných míst, např.

```
printf("%5d\n", 12345);
printf("%5d\n", 12);
printf("%05d\n", 12);
```

vypíše

```
12345
  12
00012
```

– pokud bychom chtěli pouze specifikovat počet desetinných míst, tak se místo "%6.2f" zadá jen "%.2f"
`printf("%.2f", 123.4); // vypíše 123.40`

3. lekce

7. Logické (pravdivostní) operátory

Pravdivostní hodnoty

- pravda (true) = jakékoliv nenulové číslo (typicky 1)
- nepravda (false) = nula

Pravdivostní operátory

- $A \parallel B$ = „A nebo B“ = výsledek je nepravdivý, pouze pokud je nepravdivá hodnota A i B
- $A \&\& B$ = „A a současně B“ = výsledek je pravdivý, pouze pokud je pravdivé A i B
- $!A$ = „negace A“ = výsledek má opačnou pravdivost, než A

8. Priorita operátorů

– udává, které operátory mají přednost před kterými

* / %	(nejvyšší priorita)
+ -	
< > <= >= == !=	
&&	(nejnižší priorita)

- operátory s vyšší prioritou se vyhodnocují dříve než operátory s nižší prioritou (patrné z pozice * a +)
- pokud mají operátory stejnou prioritu, tak se vyhodnocují zleva doprava tak, jak jsou zapsané v kódu
- pokud si nejsem jistý prioritou operátoru nebo pokud jí chci změnit, tak stačí výraz ozávkovat
- tabulka je zjednodušená

9. Větvení programu

Syntaxe 1

```
if (podmínka) {  
    ... příkazy ...  
}
```

– příkazy ve složených závorkách se provedou, pouze pokud je podmínka splněná (je true)

Syntaxe 2

```
if (podmínka) {  
    ... příkazy pro true ...  
}  
else {  
    ... příkazy pro false ...  
}
```

- pokud je podmínka splněná (je true), provedou se pouze příkazy v prvních složených závorkách
- pokud podmínka není splněná (je false), provedou se pouze příkazy z druhých složených závorek

- příkazy ve složených závorkách se nazývají **blok**
- příkazy v bloku je kvůli přehlednosti kódu dobré odsazovat o dvě mezery vpravo

16. Cykly