Úkolem je vytvořit program, který převede barvu z RGB zápisu do hexadecimálního formátu.

Na vstupu dostanete definici barvy v podobě rgb (x, y, z). x, y a z jsou celá čísla v intervalu od 0 do 255 včetně a reprezentují barevnou složku. Cílem je převést tento formát na formát začínající znakem # a následně bez mezer pro každou barevnou složku hexadecimální zápis na dvě pozice například pro hodnotu 12 vypíšete dvě hodnoty 0C. Formát vstupu a výstupu je vidět níže na ukázce práce programu.

Pokud je vstup neplatný, program to musí detekovat a zobrazit chybové hlášení. Chybové hlášení zobrazujte na standardní výstup (ne na chybový výstup). Za chybu považujte:

- vstup neobsahuje řetězec rgb,
- chybějící čárka nebo závorka,
- chybějící barevná složka,
- barevná složka není validní číslo,
- barevná složka není v rozmezí 0 až 255.

Ukázky běhu programu:

Zadejte barvu v RGB formatu:
rgb (255, 0, 0)
#FF0000

Zadejte barvu v RGB formatu:
    rgb   (0 , 255 , 0)
#00FF00

Zadejte barvu v RGB formatu:
rgb(0,0,255)
#0000FF

Zadejte barvu v RGB formatu:
rgb (127, 127, 0)
#7F7F00

Zadejte barvu v RGB formatu:
hsl (0, 127, 0)
Nespravny vstup.

Zadejte barvu v RGB formatu:
rgb (255, 0)
Nespravny vstup.

Zadejte barvu v RGB formatu:
rgb (1000, 127, 0)
Nespravny vstup.

Zadejte barvu v RGB formatu:
RGB (100, 200, 0)
Nespravny vstup.

Poznámky:

- Ukázkové běhy zachycují očekávané výpisy Vašeho programu (tučné písmo) a vstupy zadané uživatelem (základní písmo). Zvýraznění tučným písmem je použité pouze zde na stránce zadání, aby byl výpis lépe čitelný. Váš program má za úkol zobrazit text bez zvýrazňování (bez HTML markupu).
- Znak odřádkování je i za poslední řádkou výstupu (i za případným chybovým hlášením).
- Pro načítání vstupu se hodí funkce scanf. Pomocí funkce scanf lze i snadno kontrolovat přítomnost čárek, závorek a řetězce rgb.
- Nepokoušejte se načíst řetězec rgb do paměti a pak jej kontrolovat. Práce s řetězci je v C komplikovaná a náchylná k chybám. V této úloze to navíc není potřeba, kontrolu zvládne funkce scanf.
- Na výstupu zobrazte hexadecimální číslo s velkými písmeny. Použijte odpovídající formát pro funkci printf.
- Při programování si dejte pozor na přesnou podobu výpisů. Výstup Vašeho programu kontroluje stroj, který požaduje přesnou shodu výstupů Vašeho programu s výstupy referenčními. Za chybu je považováno, pokud se výpis liší. I chybějící nebo přebývající mezera/odřádkování je považováno za chybu. Abyste tyto problémy rychle vyloučili, použijte přiložený archiv se sadou vstupních a očekávaných výstupních dat. Podívejte se na videotutoriál (**Courses → Video tutoriály**), jak testovací data použít a jak testování zautomatizovat.

- Váš program bude spouštěn v omezeném testovacím prostředí. Je omezen dobou běhu (limit je vidět v logu referenčního řešení) a dále je omezena i velikost dostupné paměti (ale tato úloha by ani s jedním omezením neměla mít problém). Testovací prostředí dále zakazuje používat některé „nebezpečné funkce“ -- funkce pro spouštění programu, pro práci se sítí, ... Pokud jsou tyto funkce použité, program se nespustí. Možná ve svém programu používáte volání:

  ```
  int main ( void )
  {
      ...
      system ( "pause" ); /* aby se nezavrelo okno programu */
      return 0;
  }
  ```

  Toto nebude v testovacím prostředí fungovat - je zakázáno spouštění jiného programu. (I pokud by se program spustil, byl by odmítnut. Nebyl by totiž nikdo, kdo by pauzu „odmáčkl“, program by čekal věčně a překročil by tak maximální dobu běhu.) Pokud tedy chcete zachovat pauzu pro testování na Vašem počítači a zároveň chcete mít jistotu, že program poběží správně, použijte následující trik:

  ```
  int main ( void )
  {
      ...
  ```

```
    #ifndef __PROGTEST__
    system ( "pause" ); /* toto Progtest nevidi */
    #endif /* __PROGTEST__ */
    return 0;
}
```