Wprowadzenie do programowania w języku C
grupa RKr, wtorek 16:15-18:00
lista nr 3 *(na pracownię 29.10.2019) (wersja 2)*

Zadanie 1. [10p *(na pracowni)* lub 5p *(po pracowni)***]**

[2.5p | **A]** Napisz makra **MAXVAL_U32** oraz **MAXVAL_U64**, które rozwijają się do maksymalnych wartości, 32-bitowej bez znaku oraz 64-bitowej bez znaku. Nie przepisuj gotowej wartości np. z kalkulatora.
wskazówka: Wykorzystaj negację bitową oraz suffixy literałów dla uzyskania U32 oraz U64, tj. U, UL, ULL.
*Mogą to być typy (**unsigned long**) i (**unsigned long long**), albo (**unsigned int**) i (**unsigned long**), zależy od platformy.*

[2.5p | **B]** Napisz makro **BITWORD_8**(b7, b6, b5, b4, b3, b2, b1, b0)**,** które przyjmie jako argumenty wartości ośmiu bitów, czyli 0 lub 1 dla każdego bN, i stworzy z nich liczbę rozmiaru jednego bajta (**char**).
wskazówka: Wykorzystaj operacje przesunięcia bitowego w lewo o numer pozycji ustawianego bitu.
*Każdy bit, który jest parametrem makra, to liczba, np. **int**, której zerowy bit reprezentuje nasz bit.*

[2.5p | **C]** Napisz makro **MERGE_4BYTES**(by3, by2, by1, by0), które przyjmie jako argumenty wartości czterech bajtów, i stworzy z nich liczbę rozmiaru czterech bajtów (**int**, **long**, zależnie od platformy).
wskazówka: Wykorzystaj operacje przesunięcia bitowego w lewo o wielokrotność ósemki.
*Bajty możesz zbudować przy pomocy makra BITWORD_8(...), które napisałeś w punkcie **[B]**.*
dla ambitnych: Czy kolejność bajtów w fizycznej reprezentacji 4-bajtowej liczby jest taka sama jak podana w makrze?

[2.5p | **D]** Napisz funkcję wykonującą dzielenie liczb całkowitych, której wartość wynikowa (return-value) będzie wynikiem dzielenia. Funkcja ma posiadać dwie sygnatury używane w zależności od trybu kompilacji:
int divInteger(**int** N, **int** D) w RELEASE (32 bity przeznaczone na wynik operacji dzielenia),
long divInteger(**int** N, **int** D) w DEBUG (na dolnych 32 bitach wynik dzielenia, na górnych reszta z dzielenia),
pomimo dwóch sygnatur funkcja ma posiadać jedno ciało/implementację.
wskazówka: Wykorzystaj dyrektywy #ifdef / #ifndef / #endif, oraz standardowe makro NDEBUG.
Sztuczka polega na tym, że NDEBUG jest zdefiniowane w tybie RELEASE, a w trybie DEBUG nie jest.
Typ int ma być 4-bajtowy, typ long ma być 8-bajtowy, jeśli tak nie jest zmień/dopasuj typy.

Zadanie 2. [10p]

[5p] Napisz funkcję **void colorizeHexBytes**(**int** k), która wypisze na standardowym wyjściu liczbę k w reprezentacji szesnastkowej w taki sposób, że bajty będą miały naprzemiennie kolory niebieski i zielony.

[5p] Napisz funkcję **void colorizeBinNybbles**(**int** k), która wypisze na standardowym wyjściu liczbę k w reprezentacji dwójkowej w taki sposób, że nyble będą miały naprzemiennie kolory czerwony i fioletowy.

wskazówka: Na następnej stronie znajdziesz informacje jak kolorować tekst na platformach Linux/Unix i Windows.

Zadanie 3. [10p] Dostępne w serwisie SKOS.

makro-konstrukcje: ([p] parametryzowane [f] funkcyjne **)**

makro-deklaracja	#define VAR_DECL int x = 0 lub #define FUN_DECL int fun(int x) { return x +1;}
makro-deklaracja [p]	#define DECL(T, N, V) T N = V, gdy za T N V podstawimy int x 0, to otrzymamy int x = 0
makro-instrukcja	#define INSTR printf("\n")
makro-instrukcja [p]	#define INSTR(msg) printf("%s\n", msg)
makro-blok	#define BLOCK { ... } lub #define BLOCK do { ... } while (0), wariant drugi lepiej współpracuje ze średnikiem

makro-wartość (wszystkie poniższe rodzaje makr są przeliczane/przetwarzane w build-time, w run-time obecne są ich finalne/stałe wartości)

makro-literał	#define DEAD_MARK 0xDEAD lub #define ASCII_a 'a' , nazwana stała liczbowa
makro-wyrażenie	#define EXPR ((BUILD_DEF) & 0xFF00), obliczenia na makro-wartościach obecnych w środowisku budowania
makro-wyrażenie [p][f]	#define FUNC(a, b) (((a)*(b)) & 0xF), takie makra można składać jak zwykłe funkcje i nazywać **makro-funkcjami**

W makro-wyrażeniach (szczególnie makro-funkcjach) ważne jest kompletne nawiasowanie, które chroni przed błędami podstawień.

makro-dyrektywy:

#if / #else / #elif / #endif / #ifdef / #ifndef / #if defined / #if !defined / #error / # / ##

Plik o rozszerzeniu „.c" z rozwiązaniem zadania 1 powinien mieć taką strukturę:

```
#if 1 // lub 0, wtedy wyłączone z kompilacji zostaną wszystkie bloki opakowane w #if defined(TASK_CASE_A)
   #define TASK_CASE_A
#endif
// … kontynuacja dla B C D

#if defined(TASK_CASE_A)
   // funkcje I definicje
#endif
// … kontynuacja dla B C D

int main()
{
#if defined(TASK_CASE_A)
   // definicje I wykonywanie
#endif
// … kontynuacja dla B C D
}
```

Kolorowanie:

```
// WINDOWS
#include "windows.h"
HANDLE hConsole;
hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
/* sposób 1 */
SetConsoleTextAttribute(hConsole, 0x9 /*blue*/ 0xA /*green*/ 0xC /*red*/ 0xD /*violet*/); // wybierz 1 kolor
printf("colorized text\n");
/* sposób 2 (działa od Windows 10) */
SetConsoleMode(hConsole, ENABLE_VIRTUAL_TERMINAL_PROCESSING); // teraz można kolorować tak jak w LINUX/UNIX

// LINUX/UNIX oraz Windows10 (escape code \033 or \x1b)
printf("\033[0;34m"  "blue text\n");
printf("\033[0;32m"  "green text\n");
printf("\033[0;31m"  "red text\n");
printf("\033[0;35m"  "violet text\n");
printf("\033[0m"     "normal text\n");
```