Wstęp do programowania w języku C

Grupa TDr

Lista 5.

1. *(10 punktów w trakcie pracowni, później 5 punktów)*

Zdefiniuj typ par reszt z dzielenia przez osiem, czyli liczb naturalnych mniejszych od 8 (to przykładowo oznacza, że chcemy reprezentować 64 poprawne wartości tego typu). Ten typ może być reprezentowany za pomocą typu `int`:

```
typedef int Pair;
```

Napisz funkcję o deklaracji `Pair make_pair(int a, int b);`, która dla reszt `a` i `b` zwróci parę o elementach `a` i `b` (możesz użyć operacji bitowych).

Napisz funkcje o deklaracjach `int get_first(Pair p);` i `int get_second(Pair p);`, które dla pary `p` zwrócą odpowiednio pierwszy i drugi element pary.

Napisz funkcję o deklaracji `Pair sum_pairs(Pair u, Pair v);` zwracającą parę, której elementami są sumy modulo 8 elementów par `u` i `v`. Przykładowo sumą par (2, 5) i (2, 4) ma być para (4, 1), bo (2 + 2) % 8 = 4 i (5 + 4) % 8 = 1.

Napisz funkcję o deklaracji `bool are_pairs_equal(Pair u, Pair v);`, która sprawdza, czy pary `u` i `v` są równe. Typ `bool` pochodzi z biblioteki `stdbool.h`.

Do każdej z pięciu funkcji napisz test używający makra `assert` z biblioteki `assert.h` (5 punktów). Przykładowo instrukcja

```
assert(are_pairs_equal(
    sum_pairs(make_pair(2, 5), make_pair(2, 4)),
    make_pair(4, 1)));
```

nie spowoduje żadnego efektu, jeśli funkcje są zdefiniowane prawidłowo (czyli `assert` otrzyma wartość `true`). Jeśli zaś zmienimy np. `5` na `6`, to program zostanie przerwany z komunikatem o błędzie (bo `assert` otrzyma `false`). Pokaż, że Twoje testy przechodzą, ale potrafisz zmienić je tak, aby otrzymywać komunikat o błędzie.

Ciekawostka. Asercje są ciekawym sposobem testowania i dokumentowania kodu również dlatego, że jeśli wstawimy dyrektywę `#define NDEBUG` przed `#include <assert.h>`, to asercje zostaną usunięte w czasie kompilacji i nie będą powodować żadnego spowolnienia wykonywanego kodu.

2. *(10 punktów)*

Napisz funkcję o deklaracji `int read_decimal();`, która wczyta jedną liczbę całkowitą typu `int` i zwróci ją jako wartość funkcji (2 punkty).

Napisz funkcję o deklaracji `int trace_decimal(int);`, która wypisze jedną liczbę całkowitą podaną jako argument i jeden znak spacji po niej, a zwróci liczbę, którą otrzymała jako parametr (2 punkty).

Definicje tych funkcji umieść w pliku `decimalio.c`. Dla tego pliku utwórz odpowiedni plik nagłówkowy `decimalio.h`. Plik `decimalio.h` nie może include'ować `stdio.h`. Stworzony moduł ma działać dla następującego pliku `main.c` (4 punkty):

```
#include "decimalio.h"
#include "decimalio.h"

int main() {
    trace_decimal(
        trace_decimal(
            trace_decimal(read_decimal())
            + trace_decimal(read_decimal()))
        + trace_decimal(
            trace_decimal(read_decimal())
            + trace_decimal(read_decimal())));
    return 0;
}
```

Przykładowo efektem wywołania funkcji `main` dla wejścia `1 2 4 8` powinno być wypisanie liczb `1 2 3 4 8 12 15` (być może w innej kolejności).

W pliku `compile.txt` zapisz polecenie, którym kompilowany jest cały program z flagami podanymi na stronie kursu (2 punkty). Może to być po prostu skrypt kompilujący (choć SKOS tutaj zwykle nie przyjmuje plików o rozszerzeniu `sh`).

Jako rozwiązanie należy oddać 4 pliki: `decimalio.c`, `decimalio.h`, `main.c` (niezmieniony) oraz plik à la `compile.txt`. Mogą być skompresowane zipem.

3. *(10 punktów)*

w serwisie SKOS