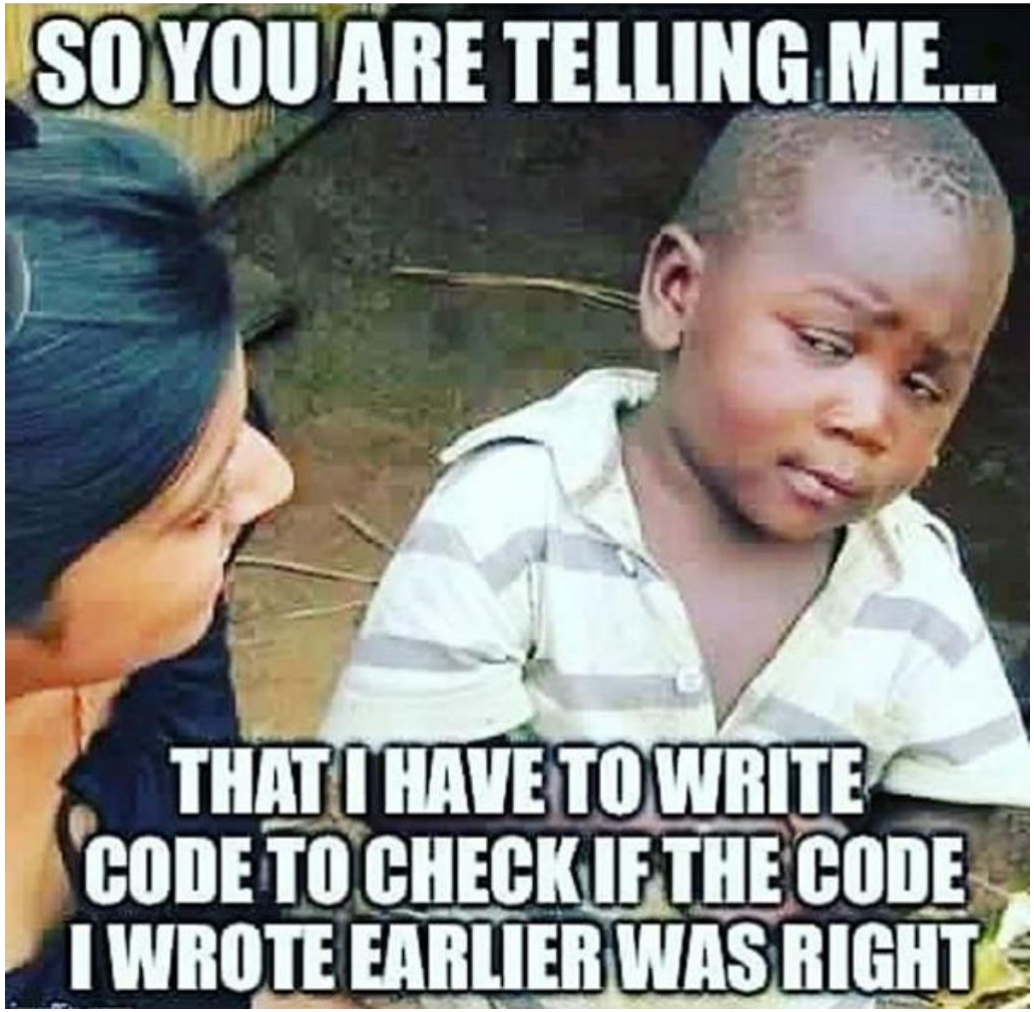
Testowanie kodu

Tak, napisany kod trzeba przetestować i w tym ćwiczeniu podejmiemy kilka kroków, które to zadanie będą miały ułatwić.

Podstawowa sprawa, to automatyzacja testów – ręczne wprowadzania wartości wejściowych i czytanie wyniku działania, żeby zastanowić się nad tym, czy jest poprawny, to nie jest dobre rozwiązanie.

Przyjrzymy się także możliwości uruchamiania (*debugging*) programu.



Zadanie małe czwarte

Kolejne małe zadanie jest przygotowaniem do przygotowania konwertera liczb zapisanych w kodzie piętnastkowym.

W ramach tych przygotowań napiszemy dwie funkcje, które będą zajmować się:

- konwersją znaku reprezentującego cyfrę na odpowiadającą jej wartość,
- konwersją wartości cyfry na reprezentujący ją znak.

Obie funkcje trzeba oczywiście przetestować, co tym razem będzie oznaczało przygotowanie stosownych funkcji sprawdzających poprawność oraz zestawu wywołań tych funkcji.

Zanim zaczniemy tworzyć rozwiązanie, sugestia zmiany w środowisku – ale środowisku domowym.

Zmiany w środowisku (domowym!)

Już w poprzednim zadaniu sugerowałem, żeby w oddawanych plikach umieszczać identyfikację autora: imię, nazwisko i nr albumu. Nie będę tego specjalnie sprawdzał, ale ułatwia mi to pracę, kiedy w jednej serii oceniam kilkanaście rozwiązań.

W oknie `Configure editor`, które otwieramy poleceniem `Settings > Editor`…, szukamy na liście po lewej stronie pozycji `Deafault code`.

W dużym pustym edytorze możemy wpisać dowolny tekst, który będzie wstawiany w każdym nowotworzonym pliku źródłowym i nagłówkowym.

Ja w obu przypadkach ustawiłem ten sam tekst:

```
// Rajmund Kożuszek (121528)
```

Każdy oczywiście wpisuje tutaj swoje dane. Za moment popatrzymy, jak to działa w praktyce.

Cyfry piętnastkowe

W odróżnieniu od poprzednich ćwiczeń, tym razem od razu tworzymy projekt docelowy: `zad_m4` i zmieniamy nazwę pliku `main.cpp` na `zad_m4.cpp`.
Mamy do zaimplementowania dwie funkcje „merytoryczne”:

```
int pentadecimal2int(char digit);
char int2pentadecimal(int value);
```

oraz dwie funkcje testujące funkcje merytoryczne:

```
bool test_pentadecimal2int(char digit, int value);
bool test_int2pentadecimal(int value, char digit);
```

Całkiem sensownie będzie zacząć od przetestowania konwersji w jedną stronę – dla ustalenia uwagi przyjmijmy, że będzie to znak na wartość.

Cyfry piętnastkowe, cd.

Do reprezentacji cyfr w kodzie piętnastkowym potrzebujemy 15 znaków. Przyjmiemy konwencję stosowaną w kodzie szestnastkowym, przy czym ostatnią dopuszczalną cyfrą będzie `'e'` (lub `'E'`).

Sprawa nie wygląda groźnie, ale będziemy mieć trzy zakresy znaków, które trzeba przełożyć na wartości:
`'0' .. '9'` – daje wartości od 0 do 9
`'a' .. 'e'` – daje wartości od 10 do 14
`'A' .. 'E'` – daje wartości od 10 do 14

Konwersja wartości na cyfrę będzie prostsza: wartości od 10 do 14 będą reprezentowane tylko przez małe litery `'a' .. 'e'`.

Pozostaje ustalić, jakie wartości powinny zwracać funkcje, kiedy dostają niepoprawny argument. Przyjmijmy, że będzie to -1 dla konwersji znak – wartość i `'x'` dla konwersji wartość – znak.

Zaczynamy

Zacznijmy od zaimplementowania w najprostszy możliwy sposób funkcji konwersji znak – cyfra i funkcji, która ma ją testować. Ta najprostsza modyfikacja, to zwracanie stałej wartości (nie ma specjalnie znaczenia, jaką wartość wybierzemy).

Dodajemy wywołanie pierwszego testu i tu naturalne jest zaczęcie od wartości brzegowej:

```
test_pentadecimal2int('0', 0);
```

Ten test, ze względu na braki w funkcji testującej, niczego ciekawego nam nie powie, ale pozwoli nam popracować nad implementacją tej funkcji, która:

1. Wywołuje `pentadecimal2int` z parametrem `digit`.
2. Zapamiętuje wartość `pentadecimal2int`.
3. Sprawdza, czy wynik jest niepoprawny i w tym przypadku wyświetla komunikat na konsoli.

Śledzenie wykonania programu

Do tej pory nie korzystaliśmy z możliwości śledzenia wykonania programu, choć budujemy programy w konfiguracji `Debug`. Pierwszą rzeczą do zrobienia (przynajmniej w laboratorium) będzie poprawienie konfiguracji debuggera. Otwieramy okno `Settings > Debugger...` i na stronie `Default` wskazujemy ścieżkę dostępu do pliku debuggera (pole `Executable path`). Warto też włączyć pierwszych pięć opcji na tej stronie.

Po tych zabiegach możemy ustawić pułapkę w funkcji testującej i dać komendę `Debug/Continue`, która rozpocznie wykonanie w trybie śledzenia.

Warto poświęcić chwilę na eksperymenty z funkcjami z menu `Debug` (i znalezienie ich przycisków na pasku narzędzi `Debugger`).

Dostarczanie rozwiązania

Rozwiązaniem zadania jest plik **`zad_m4.cpp`** (dokładnie tak ma się nazywać). Proszę złożyć tylko ten jeden plik.

Na początku pliku trzeba umieścić w komentarzu imię, nazwisko i nr albumu autora.

Pod nagłówkiem `Tydzień 4 - 24-30.10` znajdziecie Państwo zadanie zatytułowane `Cyfry piętnastkowe`. W tym zadaniu każdy z Was powinien załadować plik `zad_m4.cpp` do:

2 listopada 2022 23:59