Zadania z Języka C dla grup 2. i 5.

Zestaw IV - listopad/grudzień 2024

13. **Obliczanie liczb Fibonacciego.** Ciąg (lub *liczby*) Fibonacciego jest zdefiniowany następująco:

$$\begin{cases} F_0 = 0, \quad F_1 = 1, \\ F_n = F_{n-1} + F_{n-2} \quad \text{dla} \quad n \geqslant 2, \end{cases}$$

a zatem pierwsze 10 wyrazów ciągu to: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34. Napisz funkcję, która oblicza F_n dla zadanego n, oraz prosty program, który pobiera n jako *argument wywołania*, wywołuje wspomnianą funkcję, po czym wyprowadza wynik na standardowe wyjście. Zadanie należy wykonać na dwa sposoby:

a) funkcja wywołuje rekurencyjnie samą siebie (dwa razy na każdym poziomie) według przepisu powyżej, co pozwala otrzymać bardzo krótki i czytelny kod źródłowy;

b) funkcja jest wywoływana tylko raz, wykonuje wszystkie obliczenia z użyciem *możliwie niewielkiej* liczby zmiennych automatycznych.

Proszę porównać czasy obliczeń (*patrz* komenda systemowa `time`) dla odpowiednio dużego n; warto pomyśleć o prowadzeniu obliczeń w arytmetyce liczb zmiennopozycyjnych zamiast całkowitych.

14. **Rozkład liczby naturalnej na sumę kwadratów.** Twierdzenie sformułowane przez Lagrange'a w 1770 roku mówi, że każdą liczbę naturalną można przedstawić jako sumę czterech kwadratów liczb naturalnych[1]. Proszę napisać program, który — dla zadanego N —

a) wypisze *wszystkie* rozkłady postaci:

$$N = a^2 + b^2 + c^2 + d^2, \qquad \text{gdzie} \quad 0 \leqslant a \leqslant b \leqslant c \leqslant d;$$

b) możliwie szybko poda *jeden* rozkład postaci j.w.;

c) działając podobnie jak algorytm sita Erastotenesa, wyszuka liczby z zakresu $1, 2, \ldots, N$, których nie da się przedstawić jako sumy *mniej niż czterech* kwadratów, tzn. $a \geqslant 1$ dla wszystkich rozkładów (notacja j.w.)[2].

15. **Mrówka Langtona.** Tzw. Mrówka Langtona to ciekawy przykład automatu komórkowego wykazującego *nieredukowalną złożoność* — pomino bardzo prostych reguł działania, zachowania algorytmu nie da się przewidzieć bez wykonania dużej liczby (około $11,000$) kroków symulacji. Obliczenia przebiegają następująco: Mamy dużą szachownicę (300×300 na pewno wystarczy...), na której każde pole może

[1] Zob. np. M. Gałuszka, Delta 3/2019; Wikipedia, *Lagrange's four-square theorem*.

[2] Podpowiedź: są to liczby postaci $n = 4^\alpha(8\beta + 7)$, gdzie α i β to pewne liczby całkowite nieujemnie; a zatem najmniejszą liczbą "czterokwadratową" jest $n = 7$.

być *białe* (0) albo *czarne* (1); na początku wszystkie pola są białe. Środkowe pole (a mówiąc ściśle: *jedno z czterech środkowych*) wybieramy jako pozycję początkową mrówki; ustalam także początkowy kierunek ruchu (np. *do góry*). Dalej (*w każdym kroku*) mrówka przemieszcza się o jedno pole w wybranym kierunku, jeśli napotkane pole jest białe, zmienia jego kolor na czarny i skręca w lewo, a jeśli jest czarne — zmienia kolor na biały i skręca w prawo. Iteracje kontynuujemy tak długo, aż mrówka spróbuje wyjść poza szachownicę. (Nie polecam przyjmowania tutaj tzw. okresowych warunków brzegowych, gdyż jedynie mącą one obraz sytuacji.) Opis zachowania mrówki pomijam, aby nie psuć Państwu zabawy ;-)

Technikalia. Zadanie zasadniczo można wykonać używając terminalu tekstowego funkcji bibliotecznej `system` wywołującej komendę `clear` z powłoki; efekt będzie jednak zdecydowanie lepszy po włączeniu biblioteki graficznej. Przykładowo, można użyć biblioteki `Xlib`, działającej pod systemami UNIX/Linux/MacOS[3].

Ciekawostka dla ambitnych. Możliwe zachowania mrówki stają się zdecydowanie bardziej złożone w przestrzeni trójwymiarowej, gdzie możemy zdefiniować różne *istotnie* sekwencje zmian kierunku po zmianie koloru pola, patrz H. Hamann, *Complex Systems* **14**, 263 (2003). [W dwóch wymiarach możemy co prawda zdefiniować mrówkę "prawą" i "lewą", ale ich ewolucje będą identyczne z dokładnością do odbicia.] Jak się zdaje, podobny problem czterech wymiarach nie był jeszcze rozważany...

[3]Zob. np. program "Hello, world!" używający `Xlib`-a na stronie paulgriffits.net. Do animacji mrówki przydadzą się także funkcje `XFillRectangle()` oraz `XClearArea()`, jak również `usleep()` [z nagłówka `unistd.h`] i `XFlush()` — do opróżniania bufora graficznego.