

## ZADANIA Z JEZYKA C DLA GRUP 2., 7. I 9.

### Zestaw II - październik/listopad 2022

Dla (średnio) zaawansowanych w Języku C:

6. **Silnia rekurencyjnie i iteracyjnie.** Napisać funkcję obliczającą  $n!$  metodą rekurencyjną, tj. według wzoru

$$n! = n \cdot (n - 1)!$$

(taka funkcja wywołuje samą siebie) oraz drugą, iteracyjną:  $n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n$ . Porównać czas działania programów wykorzystujących te dwie metody dla dużych  $n$  za pomocą komendy systemowej `time`. Eleganckie wykonanie tego zadania możliwe jest po zadeklarowaniu funkcji `main` w postaci

```
int main(int nargs, char *Arg[])
```

i przyjęciu np. że pierwszy parametr wywołania (tj. `Arg[1]`) definiuje metodę liczenia, zaś drugi (tj. `Arg[2]`) liczbę  $n$ . Podczas kompilacji warto poeksperymentować z opcjami optymalizacji kompilatora `gcc`, np.

```
gcc -Olevel silnia.c -o silnia
```

gdzie `level` to liczba od 0 do 3 (więcej informacji: `man gcc`).

*Uwaga:* Chociaż naturalnym wydaje się napisanie funkcji zwracających wartości całkowite, tak wcale być nie musi. Proszę sprawdzić, dla jakiego  $n$  następuje przepełnienie zakresu liczb całkowitych i napisać ulepszone funkcje, zwracające wartości rzeczywiste podwójnej precyzji.

7. **Dwumian Newtona.** W podobny sposób napisać dwie wersje programu obliczającego symbol Newtona  $\binom{n}{k}$ . W wersji iteracyjnej przechowywać *trójkąt Pascala* w zewnętrznej, statycznej tablicy kwadratowej (*dla ambitnych:* dynamicznie alokowana tablica trójkątna). Powstałe funkcje można użyć do generacji rozkładu prawdopodobieństwa Bernoulliego

$$P_n(k) = \binom{n}{k} p^k (1-p)^{n-k},$$

dla  $k = 1, 2, \dots, n$  i np.  $p = 0.1, 0.5$  i odpowiednio dużego  $n$ . Otrzymane wyniki porównać z odpowiednimi granicznymi rozkładami Gaussa. (Odpowiednie wzory wyszukać samodzielnie.) Do wizualizacji danych można wykorzystać dowolny program typu `gnuplot`, `xmgrace`, itp.

*Uwaga:* Zamiast tablic dwuwymiarowych można użyć tablicy jednowymiarowej, w której pozycja wyznaczana jest za pomocą makr definiowanych dyrektywami `#define`.

8. **Konwersja liczb arabskich na rzymskie (i z powrotem)**. Napisać program konwertujący liczbę arabską (czytaną z klawiatury) na rzymską, oraz podobny działający w odwrotnym kierunku. Korzystając z funkcji `isdigit`, `isalpha` zdefiniowanych w pliku `<ctype.h>`) zmodyfikować program tak, aby sam rozpoznawał jaką liczbę wprowadzono i dokonywał konwersji w odpowiednim kierunku.

*Wskazówka:* W programie można wykorzystać zainicjowaną tablicę struktur:

```
struct RZYM {
    int arab;
    char *rzym;
} rz[]={1,"I"},{4,"IV"},{5,"V"},{9,"IX"},{10,"X"},
        {40,"XL"},{50,"L"},{90,"XC"},{100,"C"},
        {400,"CD"},{500,"D"},{900,"CM"},{1000,"M"};
```

zaś zakres liczb dozwolonych liczb arabskich można ustalić na  $1, \dots, 3999$ .

9. **Kalendarz**. Napisać program wczytujący datę w formacie *dzień, miesiąc, rok* i obliczający dzień tygodnia (proszę pamiętać o konieczności uwzględnienia lat przestępnych; najlepiej z uwzględnieniem kalendarzy *juliańskiego* i *gregoriańskiego*). Warto dodać opcję wypisywania pełnego kalendarza na zadany miesiąc (por. polecenie `cal`).
10. **Obliczanie liczby  $\pi$  metodą Monte Carlo**. Napisać prosty generator liczb pseudolosowych według przepisu:

$$R_{n+1} = [75 \cdot (R_n + 1) \bmod 65537] - 1.$$

Dokładnie taki generator był stosowany na popularnych w latach 80-tych komputerach *ZX Spectrum*, a generowane liczby zawierają się w przedziale  $0, \dots, R_{\max} = 2^{16} - 1$ . Można je łatwo konwertować na liczby zmiennoprzecinkowe z przedziału  $[0, 1)$  z pomocą instrukcji rzutowania: `x = (double)R / (R_MAX + 1.0)`.

Następnie, proszę wykorzystać opisany algorytm do generacji dużej liczby ( $n \sim 10^2$ ) par punktów na płaszczyźnie  $(x, y)$  należących do kwadratu  $\{0 \leq x < 1 \wedge 0 \leq y < 1\}$ . Wiedząc, że prawdopodobieństwo trafienia w koło  $\{x^2 + y^2 < 1\}$  wynosi  $\pi/4$ , wyznaczyć przybliżoną wartość liczby  $\pi$  na podstawie ułamka  $k/n$ , gdzie  $k$  oznacza liczbę trafień we wnętrze koła. Na podstawie informacji na temat *prób Bernouliego* oszacować niepewność takiego przybliżenia.

Czy obliczona w ten sposób liczba  $\pi$  zgadza się z wartością dokładną? Jeśli nie, to dlaczego? Proszę przeprowadzić opisane obliczenia dla kilku wartości  $n = 10^4, 10^5, 10^6, \dots$  (i różnych wartości początkowych  $R_0$ ) i ustalić, gdzie pojawiają się niepokojące rozbieżności. Zaproponować ulepszenie algorytmu. (W pierwszej kolejności, proszę sprawdzić co zmieni się po zastosowaniu generatora `rand()` zdefiniowanego w bibliotece standardowej `<sdtlib.h>`.)