Podstawy Programowania
Semestr letni 2022/23
Materiały z laboratorium i zadania domowe

Przemysław Olbratowski

4 kwietnia 2023

Slajdy z wykładu są dostępne w serwisie UBI. Informacje organizacyjne oraz formularz do uploadu prac domowych znajdują się na stronie `info.wsisiz.edu.pl/~olbratow`. Przy zadaniach domowych w nawiasach są podane terminy sprawdzeń.

6.2 Zadania domowe z działu Argumenty (26 kwietnia, 10, 17 maja)

6.2.1 Days: Długości miesięcy

Napisz program `days`, który przyjmuje jako argumenty wywołania rok oraz numer miesiąca i wypisuje na standardowe wyjście liczbę dni w tym miesiącu z uwzględnieniem lat przestępnych. Program załącza tylko pliki nagłówkowe `cstdlib`, `iostream` i `vector`.

Przykładowe wykonanie

```
  Linux: ./days 2000 2
Windows: days.exe 2000 2
Out: 29
```

6.2.2 Deadtime: Zagadnienie czasu martwego

Kontrolę drogową prowadzi n policjantów, każdy kontroluje jeden samochód dokładnie przez t minut, a kolejne samochody przyjeżdżają średnio co T minut. Jeżeli w chwili przyjazdu przynajmniej jeden policjant jest wolny, to samochód zostaje skontrolowany, zaś w przeciwnym razie przejeżdża swobodnie. Napisz program `deadtime`, który przyjmuje jako argumenty wywołania wartości n oraz t/T i wypisuje na standardowe wyjście prawdopodobieństwo, że przejeżdżający samochód zostanie skontrolowany. Program załącza tylko pliki nagłówkowe `cmath`, `cstdlib`, `iostream` i `vector`.

Przykładowe wykonanie

```
  Linux: ./deadtime 2 0.5
Windows: deadtime.exe 2 0.5
Out: 0.92282
```

Wskazówka Przeprowadź symulację Monte-Carlo. Losuj odstępy czasu między przyjazdami kolejnych samochodów. Rozważ dużą liczbę samochodów i policz, ile z nich zostało skontrolowanych.

6.2.3 Exact Sum: Przedział o zadanej sumie elementów

Napisz program `exact_sum`, który przyjmuje jako argument wywołania pojedynczą liczbę całkowitą, czyta ze standardowego wejścia liczby całkowite do napotkania końca pliku i wypisuje na standardowe wyjście pierwszy napotkany podciąg tych liczb o sumie zadanej argumentem wywołania. Jeżeli taki podciąg nie istnieje, program nic nie wypisuje. Zaproponuj algorytm o złożoności liniowej. Program załącza tylko pliki nagłówkowe `cstdlib`, `iostream` i `vector`.

Przykładowe wykonanie

```
  Linux: ./exact_sum 18
Windows: exact_sum.exe 18
 In: 3 7 -1 12 -5 7 10
Out: 7 -1 12
```

6.2.4 Goldbach: Hipoteza Goldbacha

Hipoteza Goldbacha mówi, że każda liczba parzysta większa od dwóch jest sumą dwóch liczb pierwszych. Przypuszczenie to nie zostało do tej pory udowodnione ani obalone. Napisz program `goldbach`, który przyjmuje jako argument wywołania liczbę naturalną i wypisuje na standardowe wyjście wszystkie mniejsze od niej liczby naturalne, których nie da się przedstawić jako sumy dwóch liczb pierwszych. Sprawdź, czy są wśród nich liczby parzyste większe od dwóch. Program załącza tylko pliki nagłówkowe `cstdlib`, `iostream` i `vector`.

Przykładowe wykonanie

```
  Linux: ./goldbach 30
Windows: goldbach.exe 30
Out: 1 2 3 11 17 23 27 29
```

6.2.5 Hanoi: Wieże Hanoi

n krążków różnej wielkości leży na stosie, mniejszy na większym. Posługując się drugim stosem jako pomocniczym należy przełożyć wszystkie krążki na trzeci stos. Można je przekładać tylko po jednym i nigdzie nie wolno położyć większego na mniejszy. Jest to tak zwany problem wież Hanoi. Krążki od najmniejszego do największego zastępujemy liczbami naturalnymi od 1 do n. Napisz program `hanoi`, który przyjmuje jako argument wywołania n i wypisuje na standardowe wyjście kolejne fazy przekładania krążków. Program załącza tylko pliki nagłówkowe `cstdlib`, `iostream` i `vector`.

Przykładowe wykonanie

```
  Linux: ./hanoi 2
Windows: hanoi.exe 2

Out: 1: 2 1
Out: 2:
Out: 3:

Out: 1: 2
Out: 2: 1
Out: 3:

Out: 1:
Out: 2: 1
Out: 3: 2

Out: 1:
Out: 2:
Out: 3: 2 1
```

6.2.6 Horner: Schemat Hornera

Rozważmy wielomian a stopnia n o współczynnikach rzeczywistych. Niech wielomian b oraz liczba r będą odpowiednio ilorazem oraz resztą z dzielenia wielomianu a przez dwumian $x - c$,

$$a = b(x - c) + r$$

Wielomian $b = b_0 + ... + b_{n-1}x^{n-1}$ oraz resztę r można obliczyć stosując kolejno wzory:

$$\begin{aligned} b_{n-1} &= a_n \\ b_{n-2} &= a_{n-1} + c \cdot b_{n-1} \\ &... \\ b_0 &= a_1 + c \cdot b_1 \\ r &= a_0 + c \cdot b_0 \end{aligned}$$

Taki sposób dzielenia wielomianu przez jednomian nazywa się schematem Hornera. Napisz program `horner`, który przyjmuje jako argument wywołania współczynnik c, następnie czyta ze standardowego wejścia współczynniki wielomianu a od zerowego do napotkania końca pliku i wypisuje na standardowe wyjście w kolejnych liniach resztę r oraz współczynniki wielomianu b. Program załącza tylko pliki nagłówkowe `iostream` i `vector`.

Przykładowe wykonanie

```
  Linux: ./horner 0.8
Windows: horner.exe 0.8
 In: 0.5 -1.2 2 3.8
Out: 2.7656
Out: 2.832 5.04 3.8
```

6.2.7 Life: Gra w życie

Populację wilków i zajęcy można w prosty sposób zasymulować na szachownicy. Każde pole może być zajęte przez wilka lub zająca albo może na nim rosnąć kapusta. Każde jest otoczone ośmioma innymi, przy czym poza szachownicą rośnie kapusta. W jednym kroku symulacji do każdego pola stosujemy następujące reguły:

- Jeżeli w otoczeniu wilka znajduje się przynajmniej jeden zając, to wilk przeżywa. W przeciwnym razie ginie i pozostawia pole kapusty.
- Jeżeli w otoczeniu zająca nie ma żadnego wilka oraz jest przynajmniej siedem pól kapusty, to zając przeżywa. W przeciwnym razie ginie i pozostawia pole kapusty.
- Jeżeli w otoczeniu pola kapusty jest więcej wilków niż zajęcy i przynajmniej jeden zając, to na polu tym rodzi się wilk. Jeżeli w otoczeniu pola kapusty są przynajmniej dwa zające i jest ich więcej niż wilków, to na polu tym rodzi się zając.

Napisz program `life` wykonujący taką symulację. Program przyjmuje jako argumenty wywołania wymiary szachownicy, rozmieszcza na niej losowo wilki oraz zające i wypisuje to ułożenie na standardowe wyjście. Następnie wczytuje ze standardowego wejścia liczbę kroków symulacji, wykonuje te kroki, wypisuje nowe ułożenie i tak dalej.

Fragment przykładowego wykonania

```
  Linux: ./life 10 20
Windows: life.exe 10 20

Out:                .  .
Out:                . . .
Out: @@             @  . .
Out:   @            @@. .
Out: .  @           @ . .
Out:  .  @          @. . .
Out: . .  @       @@. . .
Out:  . . .@@    @. . . .
Out: . . .   @@@  . . . .
Out:  . . ..      . . . .

 In: 1

Out:               .... .
Out:               . . . .
Out:              @@ . .
Out: @@             @ . .
Out: ..@           @@ . .
Out: . .@          @@ . .
Out:  . . @@     @@@ . . .
Out: . . . @     @ . . . .
Out:  . . ..@     . . . .
Out: . . . .      .. . . .
```

6.2.8 Nominals: Odliczanie kwoty w nominałach - indywidualnie

W pewnym państwie emitowane są nominały 1, 2, 5, 10, 20, 50, 100 i 200. Napisz program `nominals`, który przyjmuje jako argument wywołania kwotę bez groszy i wypisuje na standardowe wyjście dające ją w sumie nominały, od największego do najmniejszego. Program odlicza zadaną kwotę w możliwie najmniejszej liczbie nominałów dysponując nieograniczoną liczbą monet lub banknotów każdego nominału. Program załącza tylko pliki nagłówkowe `cstdlib`, `iostream` i `vector`.

Przykładowe wykonanie

```
  Linux: ./nominals 739
Windows: nominals.exe 739
Out: 200 200 200 100 20 10 5 2 2
```

6.2.9 Pascal: Trójkąt Pascala

Napisz program `pascal`, który przyjmuje jako argument wywołania liczbę naturalną n i drukuje na standardowe wyjście n pierwszych wierszy trójkąta Pascala sformatowanych jak poniżej. Szerokości kolumn są jednakowe i równe liczbie cyfr największej drukowanej liczby. Program załącza tylko pliki nagłówkowe `iomanip`, `iostream` i `vector`.

Przykładowe wykonanie

```
  Linux: ./pascal 6
Windows: pascal.exe 6
Out:           1
Out:         1   1
Out:       1   2   1
Out:     1   3   3   1
Out:   1   4   6   4   1
Out: 1   5  10  10   5   1
```

6.2.10 Permutations: Wszystkie permutacje

Napisz program `permutations`, który przyjmuje jako argument wywołania nieujemną liczbę całkowitą n i wypisuje na standardowe wyjście wszystkie permutacje liczb od 1 do n. Program załącza tylko pliki nagłówkowe `cstdlib`, `iostream` i `vector`.

Przykładowe wykonanie

```
  Latex: ./permutations 3
Windows: permutations.exe 3
Out: 1 2 3
Out: 1 3 2
Out: 2 1 3
Out: 2 3 1
Out: 3 1 2
Out: 3 2 1
```

6.2.11 Sum: Suma argumentów - grupowo

Napisz program `sum`, który przyjmuje jako argumenty wywołania nieznaną z góry liczbę wartości rzeczywistych i wypisuje na standardowe wyjście ich sumę. Program załącza tylko pliki nagłówkowe `cstdlib` i `iostream`.

Przykładowe wykonanie

```
  Linux: ./sum 0.6 -0.2 1.3
Windows: sum.exe 0.6 -0.2 1.3
Out: 1.7
```