Wojciech Myszka

Laboratorium 2: Algorytm Euklidesa

2021-03-06 09:51:21 +0100

1. Pętle

Tematem przewodnim zajęć jest zapoznanie się z dwiema podstawowymi konstrukcjami do tworzenia pętli:

1. Polecenia **while**

```
while ( warunek )
{
...
}
```

2. Polecenie **do** – **while**

```
do
{
...
} while ( warunek );
```

W miejscu gdzie pojawiają się trzy kropeczki wstawiamy polecenie/polecenia, które będą wykonywane w pętli tak długo, jak długo warunek jest **prawdziwy**.

Polecenie **while** (oraz **do** – **while**) istotnie różnią się od polecenia **if**. Na pierwszy rzut oka różnica nie jest specjalnie wielka:

```
if (m > n)
{
    m = m - n;
}
```

```
while (m > n)
{
    m = m - n;
}
```

ale istota pierwszego przykładu (**if**) jest taka: sprawdź czy m jest większe od n i jeżeli tak jest wykonaj operację odejmowania ($m = m - n$) zapisując wynik w m. W drugim przypadku (**while**) tak długo jak m jest większe od n wykonuj operację odejmowania ($m = m - n$).

W pierwszym przypadku będzie **jedno** sprawdzenie i (gdy wynik będzie pozytywny) jedno wykonanie odejmowania.

W przypadku drugim będzie cykl: sprawdzenie warunku — ewentualne wykonanie odejmowania; zawsze po wykonanym odejmowaniu sprawdzany będzie ponownie warunek. Gdy warunek nie będzie spełniony — przechodzimy do wykonywania kolejnego polecenia.

2. Algorytm Euklidesa

Wszystkie zmienne są typu **int**!

Zadania do wykonania

1. Zapoznać się z różnymi wariantami algorytmu Euklidesa zaprogramowanymi w Blockly:
   — wersja z odejmowaniem ~~opisana na pierwszym wykładzie~~
   oraz
   — wersja z resztą.
2. Napisać program realizujący algorytm Euklidesa (preferowana jest wersja „z resztą". Algorytm powinien być zaprogramowany na **co najmniej dwa sposoby**: z wykorzystanie instrukcji **while** i **do** – **while**. Ewentualnym, trzecim sposobem może być użycie pętli **for**.

2.1. Wersja „z resztą"

Dane są dwie dodatnie liczby całkowite m i n. Należy znaleźć ich *największy wspólny dzielnik*, tj. największą dodatnią liczbę całkowitą, która dzieli bez reszty zarówno m jak i n.

E1 [Znajdowanie reszty] Podziel m przez n i niech r oznacza resztę z tego dzielenia.
E2 [Czy wyszło zero?] Jeśli $r = 0$, zakończ algorytm; odpowiedzią jest n.
E3 [Upraszczanie] Wykonaj $m \leftarrow n$, $n \leftarrow r$ i wróć do kroku E1.

2.2. Wersja „z odejmowaniem"

Dane są dwie dodatnie liczby całkowite m i n. Należy znaleźć ich *największy wspólny dzielnik*, tj. największą dodatnią liczbę całkowitą, która dzieli bez reszty zarówno m jak i n.

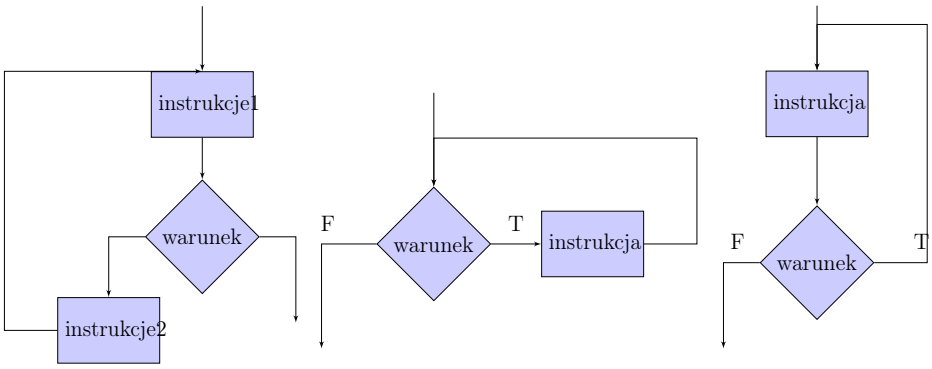
1. Jeżeli m **jest równe** n — koniec, największym wspólnym dzielnikiem jest n.
2. Jeżeli $m > n$ przyjmij $m \leftarrow m - n$; w przeciwnym razie przyjmij $n \leftarrow n - m$
3. Przejdź do kroku 1.

lub raczej, wersję zmodyfikowaną:

1. Tak długo jak $m \neq n$ powtarzaj:
   a) Jeżeli $m > n$ przyjmij $m \leftarrow m - n$; w przeciwnym razie przyjmij $n \leftarrow n - m$
2. Największym wspólnym dzielnikiem jest n.

3. Na czym polega problem?

Porównajmy uproszczone schematy blokowe
— Algorytmu E
— polecenia **while**
— petli **do**



Od lewej mamy: Algorytm E, **while**, **do**.

while zaczyna się od warunku, jak jest spełniony — wykonujemy jakieś polecenia i powtarzamy

do — najpierw wykonujemy jakieś polecenie, później sprawdzamy warunek i jeżeli jest spełniony — powtarzamy.

Algorytm E ma instrukcje przed warunkiem (jak **do**) i po warunku (jak **while**). Natomiast jego istotą jest aby operacje były wykonywane w kolejności:
instrukcje 1 → warunek → instrukcje 2 → instrukcje 1 → warunek → instrukcje 2 → . . . → warunek

W każdym przypadku korzystając ze standardowych poleceń pętlowych trzeba te „nadmiarowe” instrukcje uwzględnić.

4. Wersja PDF tego dokumentu. . .

. . . pod adresem.

Wersja: 58 z drobnymi modyfikacjami! data ostatniej modyfikacji 2021-03-06 09:51:21 +0100