Materiały dla finalistów

Malachoviacus Informaticus 2017

31 marca 2017

Wprowadzenie

Poniższy dokument zawiera opisy zagadnień, które będą niezbędne do rozwiązania zadań w drugim etapie konkursu. Polecamy dokładnie zapoznać się z dostarczonymi wyjaśnieniami. Przydatne może być również wyszukanie dodatkowych informacji na poniższe tematy na własną rękę. Zalecamy także ponowne rozwiązanie zadań z pierwszego etapu tegorocznej edycji.

Zapoznanie się i zrozumienie informacji zawartych poniżej jest częścią konkursu. Rozwiązywanie zadań na końcu każdej z sekcji *nie jest wymagane*, ale może pomóc w lepszym zrozumieniu materiału.

1 Bardziej zaawansowane procesory

Pamięć komputera składa się z ogromnej liczby *komórek*. Każda z nich jest w stanie przechować jedną liczbę *całkowitą* (dla uproszczenia przyjmijmy, że dowolnie dużą lub dowolnie małą). Komórki numerowane są kolejno od 0 – numer ten nazywamy *adresem komórki*.

Uwaga! Zwróć uwagę, że tym razem dopuszczamy także liczby ujemne jako wartości komórek!

Program wykonywany przez *procesor* jest listą kolejno wykonywanych *instrukcji*, z których każda wykonuje jedną prostą operację, jak dodawanie czy mnożenie. Każda z nich posiada adres komórki wyjściowej, w której przechowany będzie wynik operacji, oraz co najwyżej dwie wartości wejściowe: adresy komórek lub liczby.

1.1 Nowy wygląd instrukcji

Podobnie jak w zadaniu z eliminacji, stosować będziemy instrukcje wykonujące obliczenia arytmetyczne. Tym razem będą one jednak wyglądać trochę bardziej jak prawdziwe instrukcje procesora, na przykład:

- `add a, b, c`, która dodaje wartości b i c po czym umieszcza je w komórce a,

- `sub a, b, c`, która w podobny sposób wykonuje odejmowanie,
- `mul a, b, c`, która w podobny sposób mnoży liczby,

Pierwsza wartość zawsze będzie adresem komórki, natomiast pozostałe dwie wartości mogą być adresami (zapisywanymi wciąż jako `@N`) lub liczbami (`#N`).

1.2 Numerowanie instrukcji

Tym razem jednak instrukcje programu będą numerowane kolejnymi liczbami naturalnymi, a więc przykładowy program z zadania w I etapie będzie wyglądać następująco:

```
1: add 0, @1, @2
2: add 0, @0, @3
```

1.3 Skoki

Numerowanie to jest potrzebne do nowego rodzaju instrukcji: *skoków*. Skok zmienia kolejność wykonywania instrukcji jeżeli zachodzi pewien warunek. Na przykład `brz a, l`: jeżeli wartość w komórce a jest równa 0, jako następną wykonaj instrukcję l. Inne rodzaje skoków:

- `brl a, l`: wykonaj skok do instrukcji l, jeżeli wartość w komórce a jest *mniejsza niż* 0,
- `brg a, l`: wykonaj skok do instrukcji l, jeżeli wartość w komórce a jest *większa niż* 0,
- `br l`: wykonaj skok do instrukcji l, bezwarunkowo.

Uwaga! Nie musisz zapamiętywać wszystkich wymienionych wyżej nazw instrukcji, jako że są one podane jedynie dla przykładu. Istotna jest przede wszystkim ogólna zasada działania skoków.

Jeżeli warunek skoku nie zostanie spełniony, skok jest pomijany i wykonywana jest kolejna instrukcja programu. Po wykonaniu skoku nie wracamy do miejsca z którego wykonaliśmy skok, to znaczy w poniższym programie nie zostanie wykonana instrukcja 2:

```
1: brz 0, 3
2: add 0, @0, @3
3: add 0, #1, #0
```

1.4 Zakończenie działania programu

Program kończy działanie jeżeli instrukcja która powinna być wykonana jako następna nie istnieje, to jest:

- wykonana instrukcja jest ostatnią w programie i nie wykonaliśmy skoku,
- wykonany został skok do instrukcji której nie ma w programie, np. w programie składającym się z 5 instrukcji skoczyliśmy do instrukcji 7.

Wynikiem działania programu jest nadal wartość w komórce 0 po zakończeniu działania programu.

Zadania

1. IV edycja, I etap, zadanie 2: Odczytaj, przetwórz, zapisz.
2. Napisz program, który zwróci wartość bezwzględną liczby znajdującej się w komórce 1. To jest: jeżeli wartość ta jest dodatnia, zwrócona zostanie bez zmian, a jeżeli ujemna, to zwrócona zostanie liczba jej przeciwna.
3. Napisz program, który zwróci mniejszą z wartości w komórkach 1 i 2.
4. Napisz program, który obliczy wartość n-tej potęgi dwójki, gdzie n to początkowa wartość komórki 1 i jest liczbą naturalną.

2 Funkcje wyższego rzędu

Funkcja to matematyczne określenie przypisania elementów jednego zbioru elementom innego zbioru, na przykład jednej liczbie naturalnej – innej liczby naturalnej.

2.1 Argumenty i wartości

Liczbę którą podstawiamy do wzroru funkcji nazywamy *argumentem*, natomiast wynik działania opisanego we wzorze funkcji nazywamy *wartością* funkcji. Na przykład dla funkcji f opisanej wzorem $f(x) = x + 1$, x jest argumentem, a $f(x)$ lub $x + 1$ wartością.

2.2 Funkcja wielu argumentów

Funkcja może posiadać więcej niż jeden argument. Na przykład funkcja g opisana wzorem $g(a, b) = a^2 + b^2$ przyjmuje dwa argumenty. Nadal posiada jednak tylko jedną wartość dla każdej pary argumentów.

2.3 Funkcja jako argument innej funkcji

Wszystkie funkcje przyjmujące jako argumenty liczby naturalne i zwracające liczby naturalne jako wartości, możemy umieścić w jednym zbiorze, na przykład F. Jako że F jest zbiorem, możemy utworzyć także funkcję, która będzie przyjmować elementy F jako argumenty.

Rozpatrzmy na przykład funkcję d opisaną wzorem $d(f, x) = f(f(x))$, która przyjmuje dwa argumenty: funkcję f z jednej liczby naturalnej do innej, oraz liczbę naturalną x. Funkcja d obliczy wynik funkcji f dla liczby x, a następnie dla wartości funkcji $f(x)$ i ta wartość będzie wartością funkcji d dla argumentów f i x.

Funkcję d możemy zastosować do opisywania innych funkcji. Rozpatrzmy, na przykład, funkcję s opisaną wzorem $s(x) = s + 1$. Chcemy teraz opisać funkcję $h(x) = x + 2$ używając funkcji d i s. Otrzymamy wtedy $h(x) = d(s, x)$.

$$\begin{aligned} h(x) &= d(s, x) \\ &= s(s(x)) \\ &= s(x + 1) \\ &= (x + 1) + 1 \\ &= x + 2 \end{aligned}$$

Zadania

1. IV edycja, I etap, zadanie 3: Funkcje rekurencyjne
2. II edycja, I etap, zadanie 3: Rachunek lambda
3. Dana jest funkcja p, opisana wzorem $p(x) = 2x$. Używając jedynie funkcji p oraz d, opisz funkcję $c(x) = 4x$.