Egzamin na studia uzupełniające
Programowanie

Za tę część egzaminu można dostać 100 punktów. Pisząc programy, staraj się nie ograniczać do czystego kodu, ale w istotniejszych momentach podawać również komentarze, wyjaśniające Twoje intencje. We wszystkich zadaniach programistycznych liczy się również elegancja rozwiązania.

W tej części egzaminu będziemy zajmować się wyrażeniami arytmetycznymi składającymi się ze znaków `+`, `-` oraz `*`, nieujemnych liczb całkowitych oraz zmiennych. Zmienne oznaczane są literami: `a`, `b`, `c`, `x`, `y`, `z`.

Punkt 1. (**10p**) Napisz w języku C funkcję `int expr1(char *s)`, która dla napisu `s` zwraca wartość logiczna, mówiąc, czy `s` jest wyrażeniem arytmetycznym (w sensie podanej wcześniej definicji), **bez nawiasów**. Możesz założyć, że w wyrażeniu nie powinno być białych znaków (spacji, tabulacji, końców wiersza).

Punkt 2. (**30p**) Napisz w języku C funkcję `inr expr2(char *s)`, która dla napisu `s` zwraca wartość `0`, jeżeli `s` **nie jest** poprawnym wyrażeniem arytmetycznym (**z nawiasami**). W przeciwnym wypadku funkcja powinna zwracać maksymalną wielkość stosu, jaka powstanie podczas obliczenia wartości tego wyrażenia, jeżeli zostanie ono przepisane do odwrotnej notacji polskiej. [1]

W tym podpunkcie możesz otrzymać połowę punktów, jeżeli Twoja funkcja jedynie sprawdza poprawność wyrażenia. Musisz wyraźnie zaznaczyć w odpowiedzi, że wybierasz ten łatwiejszy wariant.

Punkt 3. (**10p**) Rozważamy tłumaczenie wyrażeń na program złożony z instrukcji mających postać

```
X = Val
X = Y + Z
X = Y - Z
X = Y * Z
```

zakończony instrukcją `return X`, gdzie `Val` jest wartością liczbową, `X`,`Y`, `Z` są zmiennymi z wyrażenia, bądź zmiennymi tymczasowymi (oznaczanymi `t1`, `t2`, ...). Nazwiemy taki program *programem trójkowym.*

Opisz, jak wyrażenie w odwrotnej notacji polskiej można przekształcić na program trójkowy. Czy zawsze to jest możliwe? Ile Twoja procedura potrzebuje zmiennych tymczasowych? *W tym podpunkcie nie musisz pisać programu, wystarczy precyzyjny, słowny opis.*

Uwaga: Programy w dwóch kolejnych podpunktach powinny być pisane w wybranym języku funkcjonalnym (SML, Haskell) albo logicznym (Prolog). W treściach będziemy mówić o funkcjach, Pisząc program w Prologu powinieneś zamiast funkcji o arności n tworzyć predykat o arności $n+1$, który, po zakończonym sukcesem działaniu, unifikuje ostatni argument z wynikiem funkcji. **Przedstawienie poprawnego pomysłu na rozwiązanie, mimo braku implementacji, może zaowocować przyznaniem części punktów**.

Punkt 4. (**30p**) Opisz strukturę danych z wybranego języka, która umożliwi kodowanie programów trójkowych. Dwa programy trójkowe nazwiemy równoważnymi, jeżeli dla każdego wartościowania zmiennych nietymczasowych zwracają one tę samą wartość (za pomocą instrukcji `return`).

Napisz funkcję, która bierze dwa programy trójkowe i zwraca wartość logiczną, mówiąc, czy te programy są równoważne. Efektywność nie jest istotna.

Punkt 5. (**20p**) Wielkość programu trójkowego to liczba jego instrukcji. Napisz funkcję, która bierze program trójkowy i zwraca najkrótszy program trójkowy równoważny wejściowemu, zawierający te same stałe liczbowe. Jeżeli takich programów jest więcej, może zwracać dowolnie wybrany. Nie musisz w ogóle przejmować się efektywnością rozwiązania.

[1] Przypominamy: odwrotna notacja polska dla wyrażeń bez zmiennych jest notacją beznawiasową, w której liczbę traktujemy jako polecenie położenia tejże liczby na stos, a operator jest poleceniem pobrania ze stosu argumentów, wykonaniem działania oraz wypisaniem wyniku z powrotem na stos. Wyrażenie $(1 + 2) * 3$ w ONP wygląda tak: `1 2 + 3 *`. W naszym przypadku w wyrażeniu oprócz liczb mogą występować zmienne.