

Mathematical Expression Project

Opis działania i poprawność

1. Pierwszym miejscem, w którym zaczyna się nasz program jest funkcja główna main, która inicjalizuje elementy GTK i rozstawia je w programie. Używamy jednego boxa o właściwościach wertykalnych, do którego pakujemy następnie kontrolkę txtEntry zawierającą wpisywane przez użytkownika wyrażenie, checkbox "Save space", suwak rozmiaru czcionki, przycisk "Compute", oraz gtk_drawing_area zawarty w scrolled window, co sprawia, że aplikacja dostosowuje się do każdego rozmiaru monitora.
2. Założmy, że użytkownik wpisał wyrażenie i nacisnął przycisk "Compute". Wtedy wywoływana jest funkcja btnCompute, która jest wydarzeniem wciskania przycisku.
3. "clear_expression_from_spaces":
Pierwszym etapem jest usuwanie wszystkich znaków spacji z wyrażenia. Oczywiście jest następujący fakt:
FAKT 1: Jeśli z wyrażenia usuniemy odstęp, to otrzymujemy wyrażenie równoważne.
4. Teraz sprawdzamy czy mamy do czynienia z pustym wyrażeniem. Jeśli tak, to wypisujemy, że "wyrażenie jest puste" i przerywamy.
5. "is_correct_input":
Następująca funkcja sprawdza, czy podane na wejściu znaki są zgodne z regułami wprowadzania. Tutaj jeszcze nie badamy wyrażenia.
6. "encode_unary_minuses":
Ta funkcja zakodowuje nam unarne operatory '-', żeby można je było później łatwo odróżnić od binarnych operatorów odejmowania.

Lemat 1: Dla każdego znaku '-' (jeśli jest pierwszym znakiem wyrażenia lub pierwszym znakiem po nawiasie otwierającym \Leftarrow '-' jest minusem unarnym).

" \Rightarrow " Istotnie, jeśli '-' jest pierwszym znakiem wyrażenia lub pierwszym znakiem po nawiasie otwierającym, to po jego lewej stronie nie stoi nic lub stoi nawias otwierający, zatem nie ma on lewego argumentu, czyli musi mieć jedynie prawy.

" \Leftarrow " Założmy nie wprost, że '-' nie jest pierwszym znakiem wyrażenia i nie jest pierwszym znakiem po nawiasie otwierającym, czyli po jego lewej stronie stoi operator lub zmienna.

Oczywiście gdyby stał operator, to wyrażenie byłoby niepoprawne. Jeśli zaś stoi zmienna, to taki potencjalny minus istotnie jest operatorem binarnym, sprzeczność.

Funkcja ta jest ścisłym odwzorowaniem naszego lematu. Dodatkowo ta funkcja każdy unarny minus zakodowuje jako znak '\$'.

7. "change_to_ONP":
Kolejnym krokiem jest wywołanie funkcji, która przekształca nasze wyrażenie do postaci ONP. Używany algorytm działa na stosie zdefiniowanym w pliku "stack.h" i jest dosyć trudny do zrozumienia. Jego idea bazuje na strukturze operatorów, która określa priorytety operatorów, ich łączność lewo/prawo-stronną i łączność specjalną w wyrażeniu. Funkcja ta zwróci zatem dla każdego wyrażenia jego ścisłą postać w ONP, z tym że uwzględni również unarne minusy. Zauważmy jeszcze, że funkcja ta ma złożoność liniową względem długości wyrażenia, gdyż w każdej iteracji przetwarzamy dokładnie jeden znak.

Ta operacja może się nie powieść jeśli mamy nieodpowiednie ilości nawiasów, bądź też mamy kilka kropek w zmiennej czy liczbie. Wtedy aplikacja pokaże błąd.

8. "create_tree":

Ta funkcja tworzy w skrócie skończone drzewo wyrażeniowe. Możemy rozumieć je tak, że mamy pewne węzły i liście. Węzły reprezentują operatory (unarne i binarne), a liście reprezentują stałe i zmienne. Bierzemy więc wyrażenie w postaci RPN:

- a. Dla każdej zmiennej tworzymy liść i kładziemy go na stosie.
- b. Dla każdego operatora unarnego zdejmujemy poprzedni liść, tworzymy węzeł z tym liściem i kładziemy go na stosie.
- c. Dla każdego operatora binarnego zdejmujemy dwa ostatnie liście, tworzymy z nich węzeł i kładziemy go na stosie.

W przypadku gdy brakuje jakiegoś liścia oznacza to, że operator ma niewłaściwą liczbę argumentów.

W przypadku gdy stos nie jest na końcu pusty (został jeszcze jakiś element), oznacza to, że nie użyto właściwym operatorów i ilości zmiennych.

Jest to ostatni etap sprawdzania poprawności wyrażenia.

9. Jeśli wszystko się udało, to wyrażenie zostanie dodane do listy wyrażeń i wywołane zostanie wydarzenie `gtk_widget_queue_draw(drawArea)`, które wymusi narysowanie elementu `drawArea`.

10. "do_drawing":

To tutaj odbywa się ostateczna finalizacja rysowania. Mamy dwie pomocnicze funkcje będące sercem programu.

11. "calculate_boxes":

Ta funkcja dla każdego liścia i węzła oblicza rozmiary pudełek, w które uda się je "zapakować". Mamy tutaj kilka reguł:

- a. Zmienna/Liczba:
 - i. Szerokość: szerokość wyrażenia
 - ii. Wysokość: wysokość wyrażenia
- b. '\$' (minus unarny):
 - i. Szerokość: Szerokość minusa + szerokość wyrażenia + szerokość ewentualnych nawiasów.
 - ii. Wysokość: Wysokość wyrażenia (bo minus jest na środku)
- c. '+', '-', '*':
 - i. Szerokość: Szerokość lewego wyrażenia (i jego nawiasów) + szerokość operatora + szerokość prawego wyrażenia (i jego nawiasów).
 - ii. Wysokość: maximum z wysokości wyrażenia lewego i prawego.
- d. '/':
 - i. Szerokość: Maximum z szerokości licznika i mianownika
 - ii. Wysokość: wysokość licznika + odstęp od kreski ułamkowej + wysokość mianownika.
- e. '^':
 - i. Szerokość: szerokość podstawy potęgi + szerokość wykładnika
 - ii. Wysokość: 3 / 4 wysokości podstawy + wysokość wykładnika.
- f. '_':
 - i. Szerokość: szerokość podstawy indeksowania + szerokość indeksu
 - ii. Wysokość 3 / 4 wysokości podstawy + wysokość indeksu.

12. "draw_expression":

Tutaj wykorzystujemy obliczenia poczynione przez funkcję `calculate_boxes` i wyznaczamy miejsca, w których będziemy rysować wyrażenie. Jest to funkcja bardzo podobna do `calculate_box` (jej lustrzane odbicie) z tym że wykonuje kilka dodatkowych operacji wykorzystując już obliczenia. Dla ułamka na przykład wyśrodkowuje licznik i mianownik.

13. W strukturze programu mamy jeszcze kilka wydarzeń dla przycisku, `check_buttona` czy suwaka. Generalnie ustawiają one potrzebne opcje i wywołują wydarzenie, które nakazuje na nowo narysować wyrażenia.

- a. Zaznaczenie opcji "Save space" powoduje zmianę działania funkcji `"calculate_box"` ustawiającej wysokość dla zmiennej (ustawia wysokość małych liter na małe lub duże).
- b. Zmiana suwaka rozmiaru czcionki ustawia dwie zmienne
 - i. `FontSize`
 - ii. `LineWidth`

14. Koniec działania programu.

15. Nawiasowanie wyrażeń

- a. Nawiasowanie unarnych minusów: Okazuje się, że jest to całkiem nietrywialne zadanie! Istotnie minus unarny musi być nawiasowany wtedy i tylko wtedy gdy nie jest pierwszym operatorem każdego podwyrażenia. Sprecyzujmy – przez podwyrażenie rozumiemy każde wymuszone nawiasowanie np. $(-3 + 5) * 6$, licznik i mianownik ułamka, prawa strona potęgowania (wykładnik), prawa strona indeksowania. Idea jest taka, że oznaczamy główny operator jako nawiasowany. Następnie rekurencyjnie przekazujemy zmienną `"bNewSubexpression"` zgodnie z zasadami powyżej, czyli w pewnych przypadkach wartość `true`, w pewnych przekazujemy dalej rekurencyjnie, a w pewnych zależy to od nawiasowania.