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 własnościach 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 dostosuwuje się do każdego rozmiaru monitora.
2. Załóżmy, 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. Oczywistym jest następujący fakt:
   FAKT 1: Jeśli z wyrażenie usuniemy odstępy, 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 <=> '-' jest minusem unarnym).
   "=>" 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.
   "<=" Załóżmy 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 reprezntują operatory (unarne I binarne), a liście reprezentują stałe I zmienne.
   Ta funkcja jest również bardzo ważna, ponieważ jest ostanim etapem nie tylko tworzenia, ale również sprawdzania wyrażenia. To tutaj dowiemy się, czy nie ma jakichś błedów w operatorach.
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: Szerkość 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 wysokośći wyrażenia lewego I prawego.
    d. '/':
        i. Szerokość: Maximum z szerokości licznika I mianownika
        ii. Wysokość: wysokość licznika + odstępy od kreski ułamkowej + wysokość mianownika.
    e. '^':
        i. Szerokość: szerkość podstawy potęgi + szerokość wykładnika
        ii. Wysokość: 3 / 4 wysokości podstawy + wysokosc wykladnika.
    f. '_':
        i. Szerokość: szerokość podstawy indeksowania + szerkość indeksu
        ii. Wysokość 3 / 4 wysokości podstawy + wysokosc 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.

Jakub Kuczkowiak - 11.02.2017