TKOM projekt

Arkusz kalkulacyjny

Opis programu:

Program jest prostym arkuszem kalkulacyjnym, udostępniającym opcję tworzenia własnych makr – czyli funkcji służacych do automatyzowania obsługi danych w arkuszu.

Przy tworzeniu makr użytkownik ma dostęp do całej puli komórek w arkuszu i może się do nich bezpośrednio odwoływać. Makra użytkownik może zarówno wczytywać z pliku, jak i wpisywać w odpowiednim oknie programu, po czym je wykonywać. Przy próbie wykonania nieprawidłowego makra, użytkownikowi w odpowiednim oknie zostanie wyświetlona informacja o błędzie, oraz jego powód.

W komórkach arkuszu mogą być zapisywane zarówno liczby (*int*), jak i ciągi znakowe (*string*) ale w makrach przy odczycie z komórki używane mogą być jedynie te z zapisanymi liczbami (lub puste). Pusta komórka odpowiada komórce liczbowej równej 0, a jej użycie w wyrażeniu warunkowym zwraca wartość *false* (wartość większa od 0 uznawana jest jako wartość logiczna *true*). Nowo powstałe zmienne w makrze, jeśli nie zostały przypisane im wartości, przyjmują zawsze wartość 0. W komórkach arkuszu mogą być również zapisane wyrażenia korzystające z zależności pomiędzy komórkami. Do komórki można przypisać także formułę, bazującą na zależnościach między komórkami, bądź funkcji sumowania, która jest na bieżąco aktualizowana.

Formuła w komórce poprzedzona musi być bezpośrednio znakiem ‘=’. Przy obliczaniu wartości dla formuły w danej komórce możliwe jest użycie funkcji sumowania po wierszach, bądź kolumnach, wartość sumy uzyskiwana jest poprzez użycie w formule wyrażenia `SUM( [row1, col1] - [row2, col2] )` przy czym kolejność komórek nie ma znaczenia. Komórki są sumowane od komórki o najmniejszym wierszu/kolumnie do komórki o największym wierszu/kolumnie włącznie.

W makrach dostępne są zmienne typu liczbowego (*int*) pod nazwą INT, zaś wskazania na komórki arkusza realizowane są poprzez nawiasowanie kwadratowe *[rząd, kolumna]* (np. [2, 3] – komórka arkusza z rzędu 2, kolumny 3). Zarówno rząd jak i kolumna w komórkach arkusza zaczynają się od 1, odwołanie się do komórki [0, 0] bądź z zakresu poza arkuszem będzie skutkować błędem.

Przykłady konstrukcji językowych makr:

1. Makro wypełniające początkowe 10 komórek arkusza w pierwszym rzędzie kolejnymi liczbami zaczynając od 1:

```
{
	INT cellNum = 1;
	WHILE ( cellNum < 11 ) {
		[cellNum, 1] = cellNum;
		cellNum = cellNum + 1;
	}
}
```

2. Makro pobierające z pierwszej komórki liczbę jeśli jest większa od 10 (i nie jest ciągiem znaków), mnożące je przez liczbę z drugiej komórki i zapisujące wynik do trzeciej komórki, w tym samym rzędzie.

```
{
	INT cellRow = 1;
	INT num = [cellRow, 1];
	IF ( [cellRow, 1] > 10 ) {
		num = num * [cellRow, 2];
		[cellRow, 3] = num;
	}
}
```

Tokeny:

“INT”, “IF”, “ELSE”, “BREAK”, “WHILE”, “[”, “]”, “(”, “)”, “{”, “}”, “;”, “,”, “=”, “!”, “+”, “-”, “*”, “/”, “%”, “|”, “&”, “==”, “!=”, “<”, “>”, “<=”, “>=”

Gramatyka:

body = “{” { ifStatement | whileStatement | initStatement “;” | assignStatement “;” | “BREAK” “;” } “}”

ifStatement = “IF” “(” condition “)” body [“ELSE” body]

whileStatement = “WHILE” “(” condition “)” body

initStatement = “INT” id [assignmentOp expression]

assignStatement = (id | cell) assignmentOp expression

expression = multiExp { addOp multiExp }

multiExp = primaryExp { multiOp primaryExp }

primaryExp = [unaryOp] (parentExp | id | cell | number)

parentExp = “(” expression “)”

condition = andCond { orOp andCond }

andCond = comparisonCond { andOp comparisonCond }

comparisonCond = primaryCond [comparisonOp primaryCond]

primaryCond = [unaryOp] expression

addOp = “+” | “-”

multiOp = “*” | “/” | “%”

assignmentOp = “=”

unaryOp = “!”

orOp = “|”

andOp = “&”

comparisionOp = “<” | “>” | “<=” | “>=” | “==” | “!=”

number = [“-”] digit { digit }

cell = “[” expression “,” expression “]”

id = letter { digit | letter }

digit = “0”..“9”

letter = “a”.. “z” | “A”.. “Z”

Wymagania funkcjonalne:

- Parsowanie i wykonywanie poprawnych makr zapisanych w oknie aplikacji
- Możliwość zapisania danego makra do pliku oraz wczytania makra z pliku
- Możliwość modyfikowania komórek poprzez ich wybór w programie i wpisywanie do nich wartości bezpośrednio
- Możliwość modyfikowania komórek poprzez makra, zarówno czytania z komórek (tylko komórek z danymi liczbowymi lub pustych), jak i zapisywania danych do komórek (typ komórki obojętny)

Wymagania niefunkcjonalne:

- W przypadku błędu w danym makrze – wypisanie w oknie aplikacji informacji o błędzie tj. lokalizacja, typ błędu, etap analizy makra, przy którym wystąpił błąd
- Możliwość poprawienia makra, w którym wystąpił błąd w oknie aplikacji i próba jego ponownego uruchomienia
- W przypadku wystąpienia błędu podczas wykonywania makra, arkusz nie jest cofany do stanu sprzed próby wykonania makra

Obsługa błędów:

- Błędy przy parsowaniu makra wyświetlane są w przeznaczonym do tego oknie wewnątrz aplikacji, umożliwiona jest dalsza edycja makra i jego ponowna próba uruchomienia
- Błędy przy wykonaniu makra (np. wyjście poza przestrzeń komórek dostępnych w arkuszu) powodują przerwanie wykonywania makra i wyświetlenie informacji o danym wyjątku w przeznaczonym do tego oknie wewnątrz aplikacji, umożliwiona jest dalsza edycja makra i jego ponowna próba uruchomienia, jednakże zmiany wprowadzone przez dane makro do czasu wystąpienia błędu nie są cofane

Budowa programu:

Program jest aplikacją z interfejsem graficznym, napisaną w języku C++ przy użyciu bibliotek Qt. Program wywołuje moduły do analizy makr w 2 sytuacjach: należy obliczyć wartość dla formuły w komórce, która traktowana jest jako *expression*, nie jako całe makro; należy wykonać makro dla całego arkusza, makro może składać się z wielu *body*, które są od siebie niezależne.

Aplikacja umożliwia:

- wszystkie bezpośrednie operacje na komórkach arkusza
- zapisywanie/wczytywanie makr do/z pliku
- wywoływanie modułów do obsługi makra

Modułami obsługującymi makra (zarówno obliczanie wartości formuły jak i makra) będą zaś:

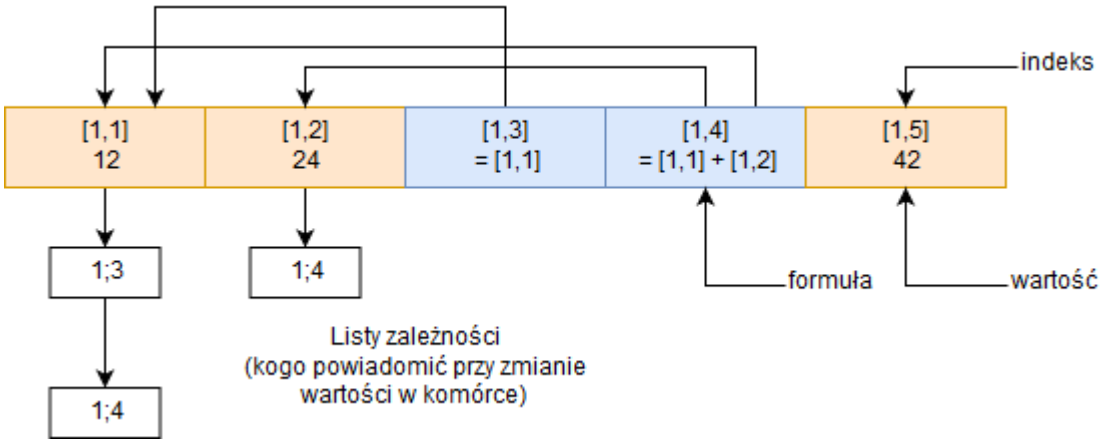
- moduł leksera – odpowiedzialny za analizę leksykalną, wykrycie błędnych znaków w kodzie, zbyt dużych liczb bądź zbyt długich identyfikatorów zmiennych
- moduł parsera – odpowiedzialny za analizę składniową, wykrycie niezadeklarowanych identyfikatorów, brak unikalnych identyfikatorów zmiennych oraz wykonanie kodu bezpośrednio po sparsowaniu, przy czym odpowiada również za wykrycie błędów przy wykonaniu kodu

Komunikacja pomiędzy komórkami:

Każda z komórek posiada listę zależności, w której utrzymuje identyfikatory komórek (jako ‘<row>;<column>’), które korzystają z jej wartości, czyli tych, które przy zmianie wartości należy powiadomić.

Ponieważ takie zmiany mogą pociągać za sobą zmiany kolejnych komórek, obliczanie zależności jest wykonywane rekurencyjnie na listach zależności danej komórki.

W programie uwzględniono możliwość istnienia zależności cyklicznych, dzięki rekurencyjnemu sprawdzeniu komórek, przy wystąpieniu zależności cyklicznych zarówno bezpośrednich jak i pośrednich, użytkownik jest o tym informowany, a wszystkie komórki, które posiadały zależność do danej, nieprawidłowej komórki, są o tym powiadamiane, po czym usuwają jej identyfikator.



Rysunek 1 - zależności pomiędzy komórkami

Testowanie programu:

Testowanie wykonywania się makr, polega na załadowaniu testowych makr z pliku, po czym uruchomieniu ich w aplikacji. Jeśli wyniki są zgodne z oczekiwaniami to znaczy, że makro wykonało się poprawnie, jeśli w makrze występują błędy, są one wyświetlane w odpowiednim oknie aplikacji.

Testowanie zależności pomiędzy komórkami testowane jest poprzez uruchomienie odpowiedniej funkcji w arkuszu (*ctrl + t*), która wpisuje do komórek odpowiednie formuły, po czym testuje ich poprawność.