Marek Moraczyński Projekt TKOM

Opis struktur binarnych – temat projektu

Język deklaratywny opisujący struktury binarne (z możliwą dokładnością do pojedynczych bitów). Projekt powinien umożliwiać zakodowanie i zdekodowanie dowolnej opisanej struktury i prezentację jej w wybranym formacie. Powinno być możliwe zdefiniowanie pól zależnych - np. pole Length i Contents.

Opis zakładanej funkcjonalności

- kodowanie i dekodowanie struktur w języku C# do/z postaci binarnej (główne zastosowanie – efektywne przesyłanie danych przez sieć)
- definiowanie typów:
    - liczb całkowitych
    - tablic
    - kontenerów dla innych typów (struct)
- instrukcje warunkowe w celu warunkowej definicji typu
- określenie rozmiaru liczb całkowitych w bitach
- podawanie parametrów struktur
- określanie wielkości tablicy poprzez podanie liczby lub jako parametr struktury
- analiza leksykalna
- analiza semantyczna
- zwrócenie miejsca niepoprawności z gramatyką

Opis realizacji programu

Język C#. Środowisko Visual Studio 2015. Główny program będzie zrealizowany w postaci biblioteki .dll. Możliwe będzie również wejście w postaci pliku jako arguemnt wywołania do analizy pod względem semantycznym i leksylanym. W programie będą znajdować się dwie podstawowe klasy dotyczące funkcjonalności programu: kodera i dekodera. Klasy te będą posiadały interfejs, aby w przyszłości ICoder, IDecoder. Kodowanie będzie zwracać wynikową tablicę bajtów. Dekodowanie odwrotnie będzie przyjmować tablicę bajtów i zwracać zdekodowaną strukturę. Jeżeli chodzi o samo przetwarzanie tekstu projekt będzie składał się z klas analizatora leksykalnego i semantycznego.

Opis testów

Przykładowe funkcjonalności do przetestowania:

1. Testowanie długości zwracanej tablicy po zdekodowaniu, z dokładnością do bajtów.
2. Testowanie poprawności kodowania.
3. Testowanie poprawności dekodowania.
4. Poprawność analizatora leksykalnego.
5. Poprawność analizatora semantycznego.

Opis języka

```
struct T(int parameter1,int  parameter2, bool boolParameter)
begin
        int zmienna0 size 7;
        int zmienna1[parameter2] size 17;
        int zmienna2[parameter1] size 10;
        if (boolParameter) then
        begin
                int zmienna3 size 10;
        end
        else
        begin
                int zmienna4[10] size 7;
        end
end

main struct()
begin
        T nazwaZmiennejWlasnegoTypu (10, 2, false);
        int zmienna3[10] size 7;
end
```

Gramatyka

Tokeny

Słowa kluczowe

begin – rozpoczęcie bloku określonego przez struct

end – zakończenie bloku określonego przez struct

struct – specjalny typ, który może zawierać w sobie typy podstawowe

int – typ podstawowy określający zmienną całkowitą

size – rozmiar zmiennej liczbowej w bitach

if – początek wyrażenia warunkowego

then - następuję po warunku w wyrażeniu warunkowym

else – alternatywna ścieżka wyrażenia warunkowego

main – deklaracja głównej struktury, która zawiera w sobie pozostałe

true/false – wartości wyrażeni regularnych

Identyfikator

Unikalne id dla zmiennej czy parametru, które złożone jest z liter i cyfr począwszy od litery.

Wartości

Liczby lub wartości zmiennych logicznych (true/false).

Składnia zdefiniowana w BNF:

<digit>::= ‘0’ | ‘1’ | … | ‘9’
<character>::= ‘a’|’b’|’c’|…|’z’|’A’|…|’Z’
<endMarker > ::=‘;’
<beginOfBlock> ::=‘begin’
<endOfBlock> ::= ‘end’
<id> ::= <character> { <character> | <digit> }
<number> ::= <digit> {<digit>}
<intType> ::= ‘int’
<boolType> ::= ‘bool’
<structure> ::= ‘struct’
<size> ::= ‘size’
<boolValue> ::= ‘true’ | ‘false’
<intValue> ::= <number>
<parameterValue> ::= <boolValue> | <intValue>
<ownTypeName> ::= <id>
<ownTypeDeclaration> ::= <ownTypeName> <id> <listOfParametersValues> <endMarker>
<ownTypeDefinition>::=<structure><ownTypeName><listOfParameters><beginOfBlock>
(<typesDeclarations> | <ifStatement>) {(<typesDeclarations> | <ifStatement>)}
<endOfBlock>
<ifStatement>::=‘if’ ‘(‘ <boolValue> | <parameter_id> ‘)’ ‘then’ <beginOfBlock>
<typesDeclarations> <endOfBlock> ‘else’ <beginOfBlock><typesDeclarations><endOfBlock>
<intTypeDeclaration> ::= <intType> <id> <size><number><endMarker>
<arrayIntTypeDeclaration> ::= <intType> <id> ‘[‘<number> |< parameter_id> ‘]’
<size><number><endMarker>

<arrayOwnTypeDeclaration> ::= <ownTypeName> <id> ‘[‘<number> |< parameter_id> ‘]’
<endMarker>
<parameter_id> ::= <id>
<parameter> ::= (<intType> | <boolType>) <parameter_id>
<listOfParameters> ::= ‘(‘{<parameter> {‘,’ <parameter> } }’)’
<listOfParametersValues> ::= ‘(‘ {<parameterValue> { ‘,’ <parameterValue>}} ‘)’
<typeDeclaration> ::= <intTypeDeclaration> | <arrayIntTypeDeclaration> |
<arrayOwnTypeDeclaration> | <ownTypeDeclaration>
<typesDeclarations> ::= <typeDeclaration> {<typeDeclaration>}
<mainStructure> ::= ‘main’ <structure> ‘(‘ ‘)’ <beginOfBlock>
<typesDeclarations><endOfBlock>