Zadanie 1.

Napisać analizator zgodności typów dla podzbioru języka Pascal. Należy założyć, że w podzbiorze występują zmienne trzech typów: *integer*, *real* i *boolean*. W języku dostępny jest zestaw operatorów, jednak nie każdego operatora można użyć do każdych argumentów i tak:

- '+', '*' mogą zostać użyte dla wyrażeń typu *integer* i *real*, i - jeśli co najmniej jeden z operandów jest rzeczywisty wynik jest także rzeczywisty,
- *div* i *mod* można użyć tylko do liczb całkowitych,
- *and* i *or* mogą być użyte dla wyrażeń typu *integer* i *boolean* ale nie jednocześnie. Jeśli oba operandy są typu całkowitego wynik jest także typu całkowitego, jeśli oba operandy są typu *boolean* to wynik jest także typu *boolean*
- >, < i = mogą zostać zastosowane do argumentów typu *integer* i *real* (w dowolnej ich kombinacji), oraz *boolean* ale nie można łączyć *boolean* z *integer* ani *real*.

Wejście rozpoczyna się od sekwencji deklaracji o postaci:
var id : *type* ;
Po deklaracjach następują wyrażenia, jedno w jednej linii. Dla każdego wyrażenia należy określić czy jest ono poprawne i wypisać „OK" jeśli jest i „Error" jeśli nie jest.
Przykład:

Wejście	`var a : integer;`
	`var a1 : integer;`
	`var b : real;`
	`var b1 : real;`
	`var c : boolean;`
	`var c1 : boolean;`
	`a + a1 * a * b`
	`a and a1`
	`b and a`
	`a < b`
	`b < c`

Wyjście	`OK`
	`OK`
	`Error`
	`OK`
	`Error`

Zadanie 2.

Napisać analizator zgodności typów dla podzbioru języka C. Należy założyć, że w języku występuje dwa możliwe typy danych: liczby całkowite *int* i liczby zmiennoprzecinkowe *double*. W języku występują również operatory, które zastosować można tylko do wyrażeń określonego typu i tak:

- lub - '||' i i - '&&' można zastosować do wyrażeń całkowitych
- '+' i '-' do dowolnych wyrażeń

Jeżeli w wyrażeniu chociaż jeden z argumentów jest typu *double* to wynik wyrażenia jest również typu *double*.
Wejście zaczyna się od sekwencji deklaracji. Po deklaracjach rozpoczynają się wyrażenia po jednym w linii. Wyrażenia mogą być ujęte w nawiasy. Dla każdego wyrażenia należy określić czy jest ono poprawne w sensie zgodności typów. Jeżeli jest poprawne należy wyprowadzić odpowiedź "OK" w przeciwnym razie "Error".
Przykład:

Wejście	`int a, a1, b, b1;`
	`double d, k, u;`
	`a || b`
	`a + b && (a-b)`
	`a || d`

Wyjście	`OK`
	`OK`
	`Error`

Zadanie 3.

Język wejściowy opisany jest następującą gramatyką:

```
S  : while E do S
   | for ( CE ; E ; CE ) do S
   | id= E
   ;
CE : E ',' CE
   | E
   ;
E  : E '+' E
   | E '-' E
   |  E '*' E
   |  E '/' E
   | E '<' E
   | E '=' E
   |  E '>' E
   | num
   | id
   ;
```

Należy napisać generator kodu pośredniego dla powyższego języka.

Przykład:

Wejście	`while a < b do`
	`  for(i=1,j=2;i<3;i++,j++)`
	`  do c=d+z`

Wyjście	`L1:  if a < b goto L1next`
	`  i := 1`
	`  j := 2`
	`L2:  if i < 3 goto L2for`
	`  goto L2next`
	`L2for:`
	`  t0 := d + z`
	`  c := t0`
	`  t1:=i + 1`
	`  i := t1`
	`  t2 := j + 1`
	`  j := t2`
	`  goto L2`
	`L2next:`
	`L1next:`

Zadanie 4.

W pliku wejściowym znajduje się ciąg wywołań funkcji *printf* języka C. Należy napisać translator dokonujący przekładu wywołań *printf* na wywołania funkcji *write* języka Pascal. Należy objąć w miarę możliwości wszystkie specyfikatory formatu języka C.
Przykład:

Wejście	`printf("%2,3d\n%f%f\n",i,j,k);` `printf("%d\n",m);`
Wyjście	`write(i:2:3,#13#10,j,k,#13#10);` `write(m);`

Zadanie 5.

Niech dany będzie język programowania zbliżony do języka C, w którym zdefiniowane jest *wywołanie funkcji* (tak jak w języku C):

```
nazwa_funkcji ( lista_parametrów )
```

oraz *operator przypisania*: =

Po lewej stronie operatora przypisania mogą występować tylko zmienne, które są zadeklarowane wcześniej. Deklaracja zmiennych wygląda podobnie jak w języku C ale tutaj dopuszczalne są tylko dwa typy: `int` oraz `float`. Po prawej stronie operatora przypisania może występować natomiast dowolne wyrażenie arytmetyczne. Dopuszczalne są operatory arytmetyczne: +, -, *, /, %, ++, --, oraz nawiasy (,).

Należy zwrócić uwagę na to, że funkcje muszą mieć swoje nagłówki, w których występuje nazwa funkcji oraz typy parametrów wywołania (tak jak w języku C parametry oddzielone są przecinkami). Nagłówek funkcji musi zawsze wystąpić przed jej pierwszym wywołaniem, w przeciwnym razie (wywołanie funkcji dla której nie było zdefiniowanego wcześniej nagłówka) traktujemy jako błąd. Nagłówki funkcji pozwalają sprawdzić zgodność typów parametrów podczas wywołania tejże funkcji.

Napisz program dla generatora YACC, który przełoży kod zapisany w tym języku na kod trójadresowy oraz sprawdzi zgodność typów dla operatora przypisania i wywołań funkcji.

Zadanie 6.

Niech dany będzie język programowania zbliżony do języka C, w którym zdefiniowana jest pętla *for* (podobnie jak w języku C w nagłówku pętli for występują trzy ciągi instrukcji oddzielonych między sobą średnikami):

```
for ( instrukcje1; instrukcje2; instrukcje3)
{
                ... instrukcje języka ...
}
```

oraz *operator przypisania*: =

Po lewej stronie operatora przypisania mogą występować tylko zmienne, które są zadeklarowane wcześniej. Deklaracja zmiennych wygląda podobnie jak w języku C ale tutaj dopuszczalne są tylko dwa typy: `int` oraz `float`. Po prawej stronie operatora przypisania może występować natomiast dowolne wyrażenie arytmetyczno-logiczne. Dopuszczalne są operatory arytmetyczne: +, -, *, /, %, ++, --, operatory relacyjne <, > , <= , >=, = =, != oraz nawiasy (,).

Należy zwrócić uwagę na inicjalizowanie zmiennych podczas ich deklaracji oraz łączność operatora przypisania.

Napisz program dla generatora YACC, który przełoży kod zapisany w tym języku na kod trójadresowy oraz sprawdzi zgodność typów dla operatora przypisania.

Zadanie 7.

Niech dane będzie instrukcja `writeln` języka Turbo Pascal. Napisz program dla generatora YACC, który przełoży taką instrukcję na równoważną instrukcję `printf()` zapisaną w języku C. Należy uwzględnić jak najwięcej możliwości formatowania wyjścia stosowanych w instrukcji `writeln` (np. określanie długości pól, itp.).

Zadanie 8.

Niech dany będzie język programowania zbliżony do języka Pacal, w którym zdefiniowana jest pętla *repeat until*:

```
repeat
 .....
instrukcje języka
 .....
until wyrażenie;
```

instrukcja warunkowa `if then else`
oraz *instrukcja przypisania*: `=`

Po lewej stronie instrukcji przypisania mogą występować tylko zmienne, które są zadeklarowane wcześniej. Deklaracja zmiennych wygląda podobnie jak w języku Pascal, a więc blok deklaracji zmiennych zaczyna się po słowie kluczowym VAR. Dopuszczalne są tylko dwa typy zmiennych: `integer` oraz `real`. Po bloku deklaracji zmiennych występuje blok instrukcji rozpoczynający się słowem kluczowym BEGIN i kończący się słowem kluczowym END (w danych występuje zawsze tylko jeden blok deklaracji zmiennych i jeden blok instrukcji).
Po prawej stronie instrukcji przypisania może występować natomiast dowolne wyrażenie arytmetyczno-logiczne. Dopuszczalne są operatory arytmetyczne: +, -, *, /, operatory relacyjne <, > , <= , >=, = =, <> oraz nawiasy (,).

Napisz program dla generatora YACC, który przełoży kod zapisany w tym języku na kod trójadresowy oraz sprawdzi zgodność typów dla instrukcji przypisania.

Zadanie 9.

Dany jest „wyciąg" z programu zredagowanego w języku C, składający się z trzech następujących sekcji:
a/ ciągu deklaracji zmiennych, o dopuszczalnych typach: `integer, char, float` oraz pochodnych wskazujących,
b/ ciągu nagłówków definicji funkcji,
c/ ciągu wywołań funkcji wymienionych w sekcji b/, o parametrach mających postać wyrażeń zbudowanych ze zmiennych wymienionych w sekcji a/ oraz operatorów: typu dodawania '+' i '−', typu mnożenia '*' i '/', wyłuskania '*' oraz referencji '&'.
Każda z sekcji poprzedzona jest wierszem o treści "$$", a ich elementy (deklaracja zamiennych, nagłówek definicji funkcji, wywołanie funkcji) zapisane są w odrębnych, pojedynczych wierszach.
Zaprogramuj w YACC-u analizator zależności kontekstowych dla opisanego wyżej języka.

Wejście	`$$`
	`int a,b,x,y;`
	`int *c;`
	`char *d,e;`
	`$$`
	`char *last(string, chr, num)  char *string, chr;  int num;`
	`void poz(z, ref)  int z;  int *ref;`
	`$$`
	`last(e, d, *c)`
	`poz(a+b*x, &y)`
	`poz(a, y)`
	`poz(a, &y, b)`

Wyjście	`last(e, d, *c)  O.K.`
	`poz(a+b*x, &y)  O.K.`
	`poz(a, y)  ERROR`
	`poz(a, &y, b)  ERROR`

Zadanie 10.

Napisz w YACC-u generator kodu pośredniego dla części wykonawczych programów mini-pascalowych, w których dopuszczone jest użycie:

- wyrażeń zbudowanych ze stałych o typach `integer` i `real`, identyfikatorów oraz operatorów '+', '−', '*' i '/',
- instrukcji prostej podstawienia oraz złożonych: `begin-end`, `case` i `repeat-until`.

Przyjmij, że kod pośredni ma mieć postać kodu trójadresowego.

Przykład

Wejście	`begin`
	`  case abc of`
	`  1,2:  x:=y+1;`
	`  3..5:  begin  y:=y*2.5;`
	`  x:=y+2;`
	`  end;`
	`  else  x:=y+3;`
	`  end;`
	`  z:=x+y/2`
	`end.`

Wyjście	`  if abc=1 goto L1;`
	`  if abc=2 goto L1;`
	`  goto L2;`
	`L1: x:=y+1;`
	`  goto L3;`
	`L2: if abc>=3 goto L4;`
	`  goto L5;`
	`L4: if abc<=5 goto L6;`
	`  goto L5;`
	`L6: y:=y*2.5;`
	`  x:=y+2;`
	`  goto L3;`
	`L5: x:=y+3;`
	`L3: t1:=y/2;`
	`  z:=x+t1;`

Zadanie 11.

Dany jest „wyciąg” z programu zredagowanego w języku Turbo Pascal, składający się z dwóch następujących sekcji:

- ciągu deklaracji zmiennych, o dopuszczalnych typach: `integer, boolean, char` oraz pochodnych tablicowych,
- ciągu instrukcji podstawienia, zredagowanych przy użyciu stałych typu `integer, boolean` lub `char`, zmiennych zadeklarowanych w sekcji a/, operatorów: addytywnych ‘+’ i ‘–’, multiplikatywnych ‘*’ i ‘/’, relacyjnych ‘<‘, ‘=‘ i ‘>‘, podstawienia ”`:=`” i indeksacji ”`[]`” oraz symboli ‘`(`’, ‘`)`’, ‘`.`’, ‘`,`’ i ‘`;`’.

Każda z sekcji poprzedzona jest wierszem o treści ”`$$`”, a ich elementy (deklaracja zmiennych, instrukcja podstawienia) zapisane są w odrębnych, pojedynczych wierszach.

Zaprogramuj w YACC-u analizator zależności kontekstowych dla opisanego wyżej języka. Analizator ten winien sprawdzać poprawność indeksacji zmiennych tablicowych oraz zgodność typów operandów użytych w poszczególnych instrukcjach podstawienia.

Przykład:

Wejście:

```
$$
i: array[1..5] of array['a'..'z'] of integer;
j: array[1..6] of array['a'..'z'] of integer;
k: array[1..8] of boolean;
m,n: array[1..2,1..5,'a'..'z'] of integer;
p: array['a'..'z'] of array[1..5] of integer;
c,d,s,x,y,z; integer;
$$
k[1]:=(x+y) < z;
i[1]:=m[1,3];
i[1]:=m[1][3];
s:=j[5]['x'];
i:=p;
k[8]:= s * (c-d);
i[6]:=m[1,3];
i:=j;
s:= x * k[5] - 1;
```

Wyjście:

```
k[1]:=(x+y) < z;   O.K.
i[1]:=m[1,3];   O.K.
i[1]:=m[1][3];   O.K.
s:=j[5]['x'];   O.K.
i:=p;   TYPES MISMATCHED
k[8]:= s * (c-d);   TYPES MISMATCHED
i[6   INDEX OUT OF RANGE
i:=j;   TYPES MISMATCHED
s:= x * k[5]   TYPES MISMATCHED
```

Zadanie 12.

Dany jest fragment programu zredagowanego w języku Turbo Pascal, składający się z dwóch następujących sekcji:

- a/ ciągu deklaracji zmiennych, o dopuszczalnych typach: `integer, boolean, char` oraz pochodnych tablicowych, o zakresach definiowanych typami okrojonymi typu `integer`,
- b/ ciągu instrukcji podstawienia, zredagowanych przy użyciu stałych typu `integer, boolean` lub `char`, zmiennych zadeklarowanych w sekcji a/, operatorów: addytywnych '+' i '-', multiplikatywnych '*' i '/', relacyjnych '<', '=' i '>', podstawienia ":=" i indeksacji "[]" oraz symboli '(', ')', '.', ',' i ';' .

Każda z sekcji poprzedzona jest wierszem o treści "$$", a ich elementy (deklaracja zmiennych, instrukcja podstawienia) zapisane są w odrębnych, pojedynczych wierszach.

Zaprogramuj w YACC-u generator kodu pośredniego dla opisanego wyżej języka. Należy przyjąć, że zadany fragment programu spełnia wszystkie wymagane zależności kontekstowe (unikatowość deklaracji zmiennych, poprawność indeksacji zmiennych tablicowych, zgodność typów operandów w poszczególnych instrukcjach podstawienia). Kod pośredni ma mieć postać kodu trójadresowego.

Przykładowo, dla pliku wejściowego o postaci:

Wejście	`$$`
	`i: array[1..5] of array[1..24] of integer;`
	`k: array[1..8] of boolean;`
	`m,n: array[1..2,1..5,1..24] of integer;`
	`s,x,y,z; integer;`
	`$$`
	`i[1]:=m[1,3];`
	`k[1]:=(x+y) < z;`

Wyjście	`    t1:=1;`
	`    t2:=24-1;`
	`    t2:=t2+1;`
	`    t2:=t1+t2;`
	`    t2:=t2-1;`
	`    t3:=1;`
	`    t4:=3-1;`
	`    t5:=24-1;`
	`    t5:=t5+1;`
	`    t4:=t4*t5;`
	`    t3:=t3+t4;`
	`    t3:=t3-1;`
	`L1: i[t1]:=m[t3];`
	`    t1:=t1+1;`
	`    t3:=t3+1;`
	`    if t1<=t2 goto L1;`
	`    t6:=x+y;`
	`    t6:=t6<z;`
	`    k[1]:=t6;`

Zadanie 13.

Niech dany będzie język kontekstowy, wzorowany, w sensie semantycznym na Pascalu. Składnię bezkontekstową języka zdefiniowano następująco:

```
Program --> DeclListL [ StatList ]
DeclList -->  Decl DeclList   |  ε
Decl --> 'const' name = Val ;  | 'var' name :  Type ;
StatList --> Stat StatList  |  ε
Stat --> name := Expr
Expr--> Val | Expr + Expr | Expr * Expr | (Expr)
Val --> inum | rnum | name
Type --> 'integer' | 'real'
```

Terminalami są symbole:

```
const, var, integer, real, name, inum, rnum oraz
':='  '=' ':' '+' '*' '(' ')' '[' ']' ';'
```

Posługując się generatorem YACC zaprogramować kompilator zdań powyższego języka na kod trójadresowy, dokonujący przekładu instrukcji i sprawdzający spełnienie następujących zależności kontekstowych:

- unikatowości deklaracji stałych i zmiennych,
- używania zmiennych lub stałych w kontekście ich wcześniejszej deklaracji,
- występowania zgodności typów lewej i prawej strony w instrukcji przypisania (jak w Pascalu), przy założeniu stosownej konwersji typów w wyrażeniach.

Przykładowe poprawne dane to:

```
const a=20;
var x:real; var i:integer; var j:real;
[ x:=2.5;
  i:=(a+i)*12;
  j:=x+2*i; ]
```

Spodziewany wynik w tym przypadku:

```
x:=2.5
t0:=20+i
i:=t0*12
t1:=itor(i)          /operacja konwersji typu 'integer' do 'real'/
t2:=2*t1
j:=x+t2
```

Dane, w których występuje brak zgodności typów w trzeciej instrukcji przypisania:

```
const a=20;
var x:real; var i:integer; var j:real;
[ x:=2.5;
  j:=(a+x)*12;
  i:=x+2*i; ]
```

Zadanie 14.

Niech dany będzie język kontekstowy, opisujący deklaracje zmiennych tablicowych i ciąg odwołań do ich elementów. Składnię bezkontekstową języka zdefiniowano następująco:

```
Program --> DeclList ; IndVarList
DeclList -->  Decl DeclList   |  ε
Decl --> Type name [ Ind ]
Ind -->  num , Ind | num      /elementy 'num' oznaczają górny zakres wymiaru tablicy, dolnym jest domyślnie '1'/
IndVarList --> IndVar  IndVarList  | ε
IndVar --> name [Ind]
Type --> 'integer' | 'char'
```

Terminalami są symbole:

```
 name, num, integer, char oraz
  ',', '[', ']', ';'
```

Posługując się generatorem YACC zaprogramować analizator zdań powyższego języka obliczający adresy względne poszczególnych tablic (element typu 'char' - 1 bajt, element typu 'integer' - 2 bajty) oraz sprawdzający spełnienie następujących zależności kontekstowych:

- unikatowość deklaracji zmiennych tablicowych,
- odwoływanie się do elementu tablicy w kontekście jej deklaracji,
- poprawność odwołań do elementów tablic (indeksy elementów mieszczą się w zadeklarowanym zakresie).

Przykład 1:

Wejście	integer a[4,3] char b[2,3,4] char c[2,2]; a[1,3] c[1,1]
Wyjście	NAZWA ZMIENNEJ ADRES WZGLEDNY a 0 b 24 c 48

Przykład 2
Natomiast dane:
char x[2,2]
integer y[3,3];
y[3,4]
zawierają błędne odwołanie do elementu tablicy y.

Zadanie 15.

Niech dany będzie język kontekstowy, zawierający deklaracje zmiennych całkowitych i logicznych oraz ciągi wyrażeń logicznych. Składnię bezkontekstową języka zdefiniowano następująco:

```
Program --> DeclList  ExprList
DeclList -->  Decl DeclList   |  ε
Decl -->  name : Type ;
Ind -->  num , Ind | num
ExprList --> Expr ; ExprList  |  ε
Expr --> Expr1 = Expr1              /przyjąć priorytety operatorów jak w Pascalu/
Expr1 --> Expr1 'or' Expr2 | Expr2
Expr2 --> Expr2 'and' Expr2 | Expr3
Expr3 --> inum | 'true' | 'false' |name | ( Expr)
Type --> 'int' | 'bool'
```

Terminalami są symbole:

```
true, false, name, inum, int, bool, and, or  oraz
',', ':', '=' ';' '(' ')'
```

Posługując się generatorem YACC zaprogramować kompilator zdań powyższego języka dokonujący ich tłumaczenia na kod trójadresowy oraz sprawdzający spełnienie następujących zależności kontekstowych:

- unikatowość deklaracji zmiennych,
- używanie zmiennych w kontekście ich wcześniejszej deklaracji,
- zgodność typów operandów w wyrażeniach (operandami '=' są wartości typu 'integer').

Przykład:

Wejście:	a:int; x:int; b:bool; (a=2) or (a=3);
Wyjście:	100: if a=2 goto 103 101: t0:=0 102: goto 104 103: t0:=1 104: if a=3 goto 107 105: t1:=0 106: goto 108 107: t1:=1 108: t2:= t0 or t1

Zadanie 16.

Posługując się generatorem YACC zaprogramować kompilator tłumaczący na kod trójadresowy uproszczoną instrukcję ‘for’ z języka C++, o następującej składni bezkontekstowej:

```
ForStat --> 'for'( DecList;name = inum ;name RelOp inum; name Instr inum) ;
DecList --> 'int' name ; DeclList | 'int' name;
RelOp --> '==' | '<' | '>'
Instr  --> += | -=
```

Terminalami są symbole:

```
int, for, name, inum, oraz
';' '==' '<' '>' '+=' '-='
```

Dodatkowo, należy sprawdzić, czy zmienna sterująca w instrukcji ‘for’ została zadeklarowana.
Przykład:

Wejście:	`for (int i; i=2;i<5;i+=2) ;`

Wyjście:	`Lbeg: i:=2`
	`if i<5 goto L1`
	`goto Lnext`
	`L1: t1:=i+2`
	`i:=t1`
	`goto Lbeg`
	`Lnext:`