San José State University
Department of Computer Engineering

CMPE 152
Compiler Design
Sections 4 and 5
Fall 2017
Instructor: Ron Mak

Assignment #4

Assigned: Thursday, October 5
Due: Monday, October 23 at 11:59 pm
Team assignment, 100 points max

New built-in `complex` type

Add a new built-in **`complex`** type to Pascal for performing complex arithmetic, i.e., with imaginary numbers. This type should be similar to a record type with two **`real`** fields named **`re`** and **`im`** for the real and imaginary parts, respectively, of a **`complex`** number.

You should be able to declare **`complex`** variables like scalars. For example:

```
VAR
    z : complex;
```

Start with the book's Chapter 12 code.

Parse `complex` expressions

To assign a value to a **`complex`** variable, use the **`re`** and **`im`** fields. For example:

```
z.re := 3.14;
z.im := -8.2;
```

Allow arithmetic expressions with the operators + - * and / between two **`complex`** variables or between a **`complex`** variable and a **`real`** variable or constant. The front end should build parse trees for such expressions as usual. Your expression type checker should allow **`complex`** variables.

```
z := x + y
```

Evaluate `complex` expressions

The backend executor does all the work to evaluate **`complex`** expressions. It should use the following rules for complex arithmetic:

- $(a + bi) + (c + di) = (a + c) + (b + d)i$
- $(a + bi) - (c + di) = (a - c) + (b - d)i$
- $(a + bi)(c + di) = (ac - bd) + (ad + bc)i$
- $\frac{a+bi}{c+di} = \frac{(ac+bd)+(bc-ad)i}{c^2+d^2}$

Standard Pascal program `Complex.pas`

The following output is from executing a standard Pascal program **`Complex.pas`** which has the **`complex`** type explicitly implemented as a **`record`** type:

```
Chapter12$ Debug/Chapter12cpp execute Complex.pas

001 PROGRAM Complex;
002
003 TYPE
004     complex = RECORD
005                   re, im : real
006               END;
007
008     mystring = ARRAY[1..3] OF char;
009
010 VAR
011     x, y, z : complex;
012
013 PROCEDURE print(name : mystring; VAR z : complex);
014     BEGIN
015         write(name, ' = (', z.re:0:5, ', ', z.im:0:5, ') ');
016     END;
017
018 PROCEDURE add(VAR x, y, z : complex);
019     BEGIN
020         z.re := x.re + y.re;
021         z.im := x.im + y.im;
022     END;
023
024 PROCEDURE subtract(VAR x, y, z : complex);
025     BEGIN
026         z.re := x.re - y.re;
027         z.im := x.im - y.im;
028     END;
029
```

```
030 PROCEDURE multiply(VAR x, y, z: complex);
031     BEGIN
032         z.re := x.re*y.re - x.im*y.im;
033         z.im := x.re*y.im + x.im*y.re;
034     END;
035
036 PROCEDURE divide(VAR x, y, z : complex);
037     VAR
038         denom : real;
039
040     BEGIN
041         denom := sqr(y.re) + sqr(y.im);
042
043         z.re := (x.re*y.re + x.im*y.im)/denom;
044         z.im := (x.im*y.re - x.re*y.im)/denom;
045     END;
046
047 BEGIN {ComplexTest}
048     x.re := 3; x.im := 2;  print('  x', x);
049     y.re := 8; y.im := -5; print('  y', y);
050     add(x, y, z);          print('x+y', z);
051     writeln;
052
053     print('  x', x);
054     print('  y', y);
055     subtract(x, y, z); print('x-y', z);
056     writeln;
057
058     x.re := 4; x.im := -2; print('  x', x);
059     y.re := 1; y.im := -5; print('  y', y);
060     multiply(x, y, z);     print('x*y', z);
061     writeln;
062
063     x.re := -3; x.im := 2;  print('  x', x);
064     y.re := 3;  y.im := -6; print('  y', y);
065     divide(x, y, z);        print('x/y', z);
066     writeln;
067
068     x.re := 5; x.im := 0; print('  x', x);
069     y.re := 3; y.im := 2; print('  y', y);
070     add(x, y, z);         print('x+y', z);
071     writeln;
072
073     x.re := 5; x.im := 4; print('  x', x);
074     y.re := 2; y.im := 0; print('  y', y);
075     multiply(x, y, z);    print('x*y', z);
076     writeln;
077
078     x.re := -2; x.im := -4; print('  x', x);
079     y.re := 0;  y.im :=  1; print('  y', y);
080     divide(x, y, z);        print('x/y', z);
081     writeln;
082 END {ComplexTest}.
```

```
                82 source lines.
                 0 syntax errors.
              0.00 seconds total parsing time.
x = (3.00000, 2.00000)   y = (8.00000, -5.00000) x+y = (11.00000, -3.00000)
x = (3.00000, 2.00000)   y = (8.00000, -5.00000) x-y = (-5.00000, 7.00000)
x = (4.00000, -2.00000)   y = (1.00000, -5.00000) x*y = (-6.00000, -22.00000)
x = (-3.00000, 2.00000)   y = (3.00000, -6.00000) x/y = (-0.46667, -0.26667)
x = (5.00000, 0.00000)   y = (3.00000, 2.00000) x+y = (8.00000, 2.00000)
x = (5.00000, 4.00000)   y = (2.00000, 0.00000) x*y = (10.00000, 8.00000)
x = (-2.00000, -4.00000)   y = (0.00000, 1.00000) x/y = (-4.00000, 2.00000)

              100 statements executed.
                0 runtime errors.
```

Pascal program `ComplexBuiltIn.pas`

Your goal for this assignment is to eliminate the need for the Pascal programmer to:

- Explicitly declare **`complex`** as a record type. Instead, it will be a new built-in type.
- Write functions to perform complex arithmetic operations. Instead, your backend executor will know how to perform the operations. The executor will implement the operations by calling member functions that you write in C++ according to the complex arithmetic rules shown above.

After you implement the new built-in **`complex`** type, your new Pascal compiler and interpreter will be able to parse and execute this program **`ComplexBuiltIn.pas`**:

```
PROGRAM ComplexBuiltIn;

TYPE
    mystring = ARRAY[1..3] OF char;

VAR
    x, y, z : complex;

PROCEDURE print(expr : mystring; VAR z : complex);
    BEGIN
        write(expr, ' = (', z.re:0:5, ', ', z.im:0:5, ') ');
    END;

BEGIN {ComplexTest}
    x.re := 3; x.im := 2;  print('  x', x);
    y.re := 8; y.im := -5; print('  y', y);
    z := x + y;            print('x+y', z);
    writeln;

    print('  x', x);
    print('  y', y);
    z := x - y; print('x-y', z);
    writeln;

    x.re := 4; x.im := -2; print('  x', x);
    y.re := 1; y.im := -5; print('  y', y);
    z := x*y;              print('x*y', z);
    writeln;

    x.re := -3; x.im := 2;  print('  x', x);
    y.re := 3;  y.im := -6; print('  y', y);
    z := x/y;               print('x/y', z);
    writeln;

    x.re := 5; x.im := 0; print('  x', x);
    y.re := 3; y.im := 2; print('  y', y);
    z := x + y;           print('x+y', z);
    writeln;

    x.re := 5; x.im := 4; print('  x', x);
    y.re := 2; y.im := 0; print('  y', y);
    z := x*y;             print('x*y', z);
    writeln;

    x.re := -2; x.im := -4; print('  x', x);
    y.re := 0;  y.im :=  1; print('  y', y);
    z := x/y;               print('x/y', z);
    writeln;
END {ComplexTest}.
```

Turn on the cross-reference listing and the parse tree listing with **`-ix`** command-line options.

Tips

Examine **`wci::intermediate::symtabimpl::Predefined`** to see how the built-in types like **`integer`** and **`real`** are defined.

Examine **`wci::frontend::pascal::parsers::RecordTypeParser`** to see what information is entered into the symbol table for a **`record`** type.

In **`wci::intermediate::symtabimpl::Predefined`** , create the complex record type and enter two real fields, **`im`** and **`re`**, into its symbol table.

What to turn in

This is a team assignment. Each team turns in one assignment and each team member will get the same score. Create a zip file that contains:

- All your **`.h`** and **`.cpp`** source files. Verify that Chapter 12's **`makefile`** can compile and execute source file **`ComplexBuiltIn.pas`**.

```
make execute src=ComplexBuiltIn.pas
```

- A text file that contains the listings and output from running the above test Pascal program with the new built-in **`complex`** type.

Submit into Canvas: **Assignment #4: New `complex` type**.

Rubric

Your program will be graded according to these criteria:

Criteria	Max points
• Correct entry for **`complex`** in the global symbol table for the new built-in type.	• 20
• Parser modifications to handle the new **`complex`** type.	• 20
• Successfully parse program **`ComplexBuiltIn.pas`**.	• 20
• Correct parse trees for **`complex`** arithmetic expressions.	• 20
• Correct execution of the test program with the new built-in **`complex`** type.	• 20