భారతీయ సాంకేతిక విజ్ఞాన సంస్థ హైదరాబాద్
भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Compilers-II

CPlex

language-specification

By Group-12

INDEX

1 Introduction

1.1 Motivation

This language's inspiration stems from two distinct sources. For many years, compilers were responsible for converting high-level code into binary or assembly code. This put a program's effectiveness wholly dependent on the programmer's coding abilities. Compilers shouldn't be left behind as AI has recently advanced in all industries. More than only translators must be included in the compilers. Many studies are being conducted on the subject everywhere in the world. The more optimizations we do in the compailer, faster we get the output.

We took it as an opportunity and we have decided to build a new language called CPlex. CPlex is primarily used for Complex Numbers computations. Complex numbers are mainly used in Electrical Engineering, Signal Processing, Quantum Mechanics, Computer Graphics, Control systems etc. Many scientists make research on the areas mentioned above. This motivates us to build a programming language based on Complex Numbers.

1.2 Goal

CPlex aims to extend its capabilities to a broader, generalized domain. CPlex offers built-in support for complex numbers, facilitating arithmetic operations like addition, subtraction, multiplication, and division.

2 Data types

2.1 Computational

Datatype	**Example**
int	3, 7, 0 etc
cint	3+4i, 6+11i etc
double	3.14, 0 etc
cdouble	3.14+7i, 8+5.5i, 3.14+5.5i etc

Table 1: Data Types Table

2.2 Non-computational

Datatype	**Example**
str	”hi this compilers project”, ””, ” ” etc
bin	0 and 1

Table 2: Data Types Table

3 Operators and expressions

3.1 Precedence - Lowest to Highest

Symbol	Purpose	Associativity	Valid Operands
++	Increment	right to left	computational datatypes
- -	Decrement	right to left	computational datatypes
rem	Modulo	left to right	int, float
/	Division	left to right	computational datatypes
*	Multiplication	left to right	computational datatypes
+	Addition	left to right	computational datatypes
-	Subtraction	left to right	computational datatypes
eq, neq, <, <=, >=, >	Relational Op	left to right	computational datatypes
and, or, neg	Logical Op	left to right	bin
=	Assignment Op	right to left	initialise, assignment

3.2 Expressions

3.2.1 RHS side of expression:

Arithmatic expression should follow BODMAS rule. For example if we have an expression like a+b*c-d/e then it should be evaluated as b*c first then a+b*c then d/e and then a+b*c-d/e.

3.2.2 Unary Operators:

1. For int and double datatype the unary operators follows default rule as C language
2. For cint, cdouble datatype we can increment as our wish. For Ex for incrementing real part of a complex number a, we write a.r++;
3. For imaginary part of a complex number a we increment as a.i++;
4. For incrementing both real and imaginary part of a complex number a we write a++;

4 Lexical specifications

4.1 Reserved keywords

Reserved keywords are the keywords that have special meaning in the program and can not be used directly as a variable name. The reserved keywords which are used in Cplex are

iter	until	return	rem
eq	neq	and	or
neg	int	cint	double
cdouble	str	bin	real
img	pow	polar	conjugate
mod	arg	angle	dist
cprint	rotate	choice	alt
default	dist	get_line	is_triangle
get_centroid	get_circumcenter	get_orthocenter	get_incenter
get_excenter	get_area	get_perimeter	

4.2 Identifiers

Identifiers are case-sensitive and consist of letters, digits, and underscores. Identifiers must start with a letter, which can be followed by a sequence of letters, digits, and underscores.

Valid Identifiers:

```
counter, va_lu_e, MAXIMUM_VALUE, myFunction, x2
```

Invalid identifiers:

```
1stPlace (starts with a digit)
float (a reserved keyword)
user-name (contains a hyphen, which is not allowed)
```

4.3 Punctuation

Punctuation characters are used to separate elements and control program flow.

1. **Semicolon(;):** Every statement should end with semicolon.
2. **Comma(,):** Comma is used to seperate the arguments in function call and in iter loop
3. **Collon(:):** Collon is used to seperate the arguments from return type in function definition.
4. **Double Quotes(" "):** Double quotes are used to define string.

4.4 Special symbols

Special symbols are characters with specific meanings and purposes within the language. They are used for operators, punctuation, and other syntactical elements.

1. **Curly braces ({ }):** Curly braces are used to define the scope of the function.
2. **Square braces ([ ]):** Square braces are used to define the array.
3. **Round braces (( )):** Round braces are used to define the arguments in function call and function definition, loops, conditional statements.

4.5 Comments

Comments are used to provide explanatory notes within the source code. They are not executed by the compiler and are for the benefit of programmers. There are two types of comments Single-line, Multi-line.

```
// UNTIL END LINE
/* MULTILINE COMMENT */
```

4.6 Whitespace

White spaces refer to spaces, tabs, and newline characters that are used for formatting and separating code elements. White spaces are generally ignored by the compiler, and their primary purpose is to enhance code readability.

- Space (' ')
- Tab (\t)
- Newline (\n)

5 Declarations

5.1 Integers:

1. **int a, b, c;**
   Integers without initialization. a, b,c can be declared without initialization.

2. **int a = 5;**
   Integers with initialization. a can be declared with initializing.

5.2 Decimals:

1. **double a, b, c;**
   Decimal numbers without initialization. a, b,c can be declared without initialization.

2. **double a = 5.6;**
   Decimal numbers with initialization. a is assigned to 5.6 .

5.3 Complex declarations:

5.3.1 Complex Numbers without real part:

1. **cint a(3), b(4), c(10);**
   Integer type complex numbers. The above statements makes declarations as a = 3i, b = 4i, c = 10i.

2. **cdouble c(3.4), d(4.7), e(10.03);**
   The above statements makes declarations as c = 3.4i, d = 4.7i, e = 10.03i.

5.3.2 Complex Numbers with real part:

1. **cint a(3, 4);**
   The above statement declares a complex number a = 3 + 4i.

2. **cdouble a(3.5, 4.7);**
   The above statement declares a complex number a = 3.5 + 4.7i.

5.4 Arrays:

5.4.1 Integer Arrays:

1. **int a[10];**
   Integer array without initialization. The above statement allocates a with size of 10 where we can declare int datatype numbers.

2. **int a(23)[10];**
   Integer array with initialization. The above statement allocates a with size of 10 and are initialized to 23.

5.4.2 Double Arrays:

1. **double d[25];**
   Double array without declaration. The above statement allocates size of 25 where we can declare double datatype numbers.

2. **double d(3.8)[25];**
   Double array with declaration. The above statement allocates a with size of 25 and are initialized to 3.8 .

5.4.3 Complex Number arrays:

1. **cint a(3)[10];**
   This statement declares an array of size 10 with each value assigned to 3.

2. **cint a(3, 4)[20];**
   This statement declares an array of size 20 with each value assigned to a complex number 3 + 4i. If user wants to declare any complex number other than 3 + 4i, then he/she has to declare manually below the declaration part.

1. **cdouble a(3.4)[10];**
   This statement declares an array of size of 10 with each value assigned to 3.4 .

2. **cdouble a(3.4, 6.93)[5];**
   This statement declares an array of size of 5 with each value assigned to 3.4 + 6.93i.

5.5 Function Declaration SYNTAX:

```
FUNC_NAME RETURN_TYPE : (DTYPE VAR1, DTYPE VAR2)  {
   /*
   CODE TO BE WRITTEN HERE
   */
}
```

First we declare function name then we mention return type and then we have a Colon after that we declare arguments separated by comma. After the parenthesis we write as usual code. Parenthesis are completely optional for the arguement declaration.

6 Statements

6.1 Intializations / Assignments

- Every intialization or assignment statement in our language ends with ";".
- Every statement has an equal to symbol(=) in which LHS contains variable and RHS contains value or return value or expression . It also contains a only a variable in RHS it means we are assigning RHS variable value to LHS variable
  Examples:
- a = 2;
- b = func(a);
- c = a+b;
- d = c;

6.2 Function call

- A function cal can be inbuit function call or user function call.
- Every function call statement in our language ends with ";".
- A function call can be done by writing the function name followed by parenthesis, and in parenthesis we write arguments in the same order as that of in the function declaration. The type of arguments should be same as in function declaration
  Examples:
- func(a,b,c);
- get_centriod(a,b,c);

6.3 Return Type

- Return statement must be written in a function scope.
- Every return statement in our language ends with ";".
- A return statement contains return word followed by a value or a variable or a expression. The value or variable or expression value should be same as return type of function.
  Examples:
- return 2;
- return a;
- return a+b;

6.4 Conditionals

6.4.1 `choice(cond) {...}`

The conditionals `choice(cond) {...}` is similar to the `if` statement in C. The condition `cond` is evaluated and if it is true then the statements inside the curly braces are executed. If the condition is false then the statements inside the curly braces are not executed. A sample code snippet is shown below:

```
choice(a eq b) {
    cprint(1);
}
```

Figure 1: Output for table 'department' and k=10

6.4.2 `choice(cond) {...} default {:. }`

The conditionals `choice(cond) {...} default {:. }` is similar to the `if-else` statement in C. The condition `cond` is evaluated and if it is true then the statements inside the first curly braces are executed. If the condition is false then the statements inside the second curly braces are executed. A sample code snippet is shown below:

```
choice(a eq b) {
    cprint(1);
} default {
    cprint(0);
}
```

Figure 2: Output for table 'department' and k=10

6.4.3 `choice(cond) {...} alt(cond) {...} default {:. }`

The conditionals `choice(cond) {...} alt(cond) {...} default {:. }` is similar to the `if-else if-else` statement in C. The condition `cond` is evaluated and if it is true then the statements inside the first curly braces are executed. If the condition is false then the condition `cond` is evaluated and if it is true then the statements inside the second curly braces are executed. If the condition is false then the statements inside the third curly braces are executed. A sample code snippet is shown below:

```
choice(a eq b) {
    cprint(1);
} alt(a eq c) {
    cprint(2);
} default {
    cprint(0);
}
```

Figure 3: Output for table 'department' and k=10

6.5 Loops

6.5.1 `iter(cond1;cond2;cond3) {...}`

The loop `iter(cond1;cond2;cond3) {...}` is similar to the `for` loop in C. The condition `cond1` is evaluated and if it is true then the statements inside the curly braces are executed. After the execution of the statements inside the curly braces the condition `cond2` is evaluated and if it is true then the statements inside the curly braces are executed. This process is repeated until the condition `cond3` is true. A sample code snippet is shown below:

```
iter(i=0;i<10;i=i+1) {
    cprint(i);
}
```

Figure 4: Output for table 'department' and k=10

6.5.2 `until(cond) {...}`

The loop `until(cond) {...}` is similar to the `while` loop in C. The condition `cond` is evaluated and if it is true then the statements inside the curly braces are executed. This process is repeated until the condition `cond` is true. A sample code snippet is shown below:

```
until(i<10) {
    cprint(i);
    i=i+1;
}
```

Figure 5: Output for table 'department' and k=10

7 Built-in functions

7.1 Inbuilt complex functions:

- `real double : (cdouble c)`: Returns the real part of the complex number `c`.
- `img double : (cdouble c)`: Returns the imaginary part of the complex number `c`.
- `pow cdouble : (cdouble base,double exponent)`: Returns the complex number $(base)^{(exponent)}$. This is done by using De Moivre's formula.
- `polar void :(cdouble c)`: Prints the polar form of a complex number `c`. Given a complex number $c = a + ib$ the polar form looks $c = r(e^{i\theta})$ (Where θ is the argument of the complex number and r is the modulus of the complex number).
- `conjugate cdouble : (cdouble c)`: Returns the conjugate of the complex number `c`. Given a complex number $c = a + ib$ the conjugate looks like $c = a - ib$.
- `mod double : (cdouble c)`: Returns the modulus of the complex number `c`. Given a complex number $c = a + ib$ the modulus looks like $c = \sqrt{a^2 + b^2}$.
- `arg double :(cdouble c)`: Returns the argument of the complex number `c`. Given a complex number $c = a + ib$ the argument looks like $c = \tan^{-1}(\frac{b}{a})$.

- `angle double : (cdouble c1,cdouble c2)`: Returns the angle between the complex numbers `c1` and `c2`. Given two complex numbers $c_1 = a_1 + b_1 i$ and $c_2 = a_2 + b_2 i$ the angle between them looks like $c = \tan^{-1}(\frac{b_2 - b_1}{a_2 - a_1})$.

- `dist double : (cdouble c1,cdouble c2)`: Returns the distance between the complex numbers `c1` and `c2`. Given two complex numbers $c_1 = a_1 + b_1 i$ and $c_2 = a_2 + b_2 i$ the distance between them looks like $c = \sqrt{(a_2 - a_1)^2 + (b_2 - b_1)^2}$.

- `cprint void: (cdouble c)`: Prints the complex number `c` in the form $a + ib$.

7.2 Geometry related:

- `rotate cdouble : (cdouble c,cdouble origin,double angle)`: Returns the complex number `c` rotated by an angle `angle` about the point `origin`. The rotation is done in the counter-clockwise direction.

- `dist double :(cdouble c1,cdouble c2)`: Returns the distance between the complex numbers `c1` and `c2`. Given two complex numbers $c_1 = a_1 + b_1 i$ and $c_2 = a_2 + b_2 i$ the distance between them looks like $c = \sqrt{(a_2 - a_1)^2 + (b_2 - b_1)^2}$.

- `get_line void :(cdouble c1,cdouble c2,double *a,double *b,double *c)`: Given two complex numbers $c_1 = a_1 + b_1 i$ and $c_2 = a_2 + b_2 i$ this function prints the line $ax + by + c = 0$ passing through the points c_1 and c_2.

- `is_traingle bin :(cdouble c1,cdouble c2,cdouble c3)`: Given three complex numbers $c_1 = a_1 + b_1 i$,$c_2 = a_2 + b_2 i$ and $c_3 = a_3 + b_3 i$ this function returns true if the points c_1,c_2 and c_3 form a triangle else false.

- `get_centroid cdouble:(cdouble c1,cdouble c2,cdouble c3)`: Given three complex numbers $c_1 = a_1 + b_1 i$,$c_2 = a_2 + b_2 i$ and $c_3 = a_3 + b_3 i$ this function returns the centroid of the triangle formed by(if exists) the points c_1,c_2 and c_3.

- `get_circumcenter cdouble :(cdouble c1,cdouble c2,cdouble c3)`: Given three complex numbers $c_1 = a_1 + b_1 i$,$c_2 = a_2 + b_2 i$ and $c_3 = a_3 + b_3 i$ this function returns the circumcenter of the triangle formed by(if exists) the points c_1,c_2 and c_3.

- `get_orthocenter cdouble :(cdouble c1,cdouble c2,cdouble c3)`: Given three complex numbers $c_1 = a_1 + b_1 i$,$c_2 = a_2 + b_2 i$ and $c_3 = a_3 + b_3 i$ this function returns the orthocenter of the triangle formed by(if exists) the points c_1,c_2 and c_3.

- `get_incenter cdouble :(cdouble c1,cdouble c2,cdouble c3)`: Given three complex numbers $c_1 = a_1 + b_1 i$,$c_2 = a_2 + b_2 i$ and $c_3 = a_3 + b_3 i$ this function returns the incenter of the triangle formed by(if exists) the points c_1,c_2 and c_3.

- `get_excenter cdouble :(cdouble c1,cdouble c2,cdouble c3)`: Given three complex numbers $c_1 = a_1 + b_1 i$,$c_2 = a_2 + b_2 i$ and $c_3 = a_3 + b_3 i$ this function returns the excenter of the triangle formed by(if exists) the points c_1,c_2 and c_3.

- `get_area double :(cdouble c1,cdouble c2,cdouble c3)`: Given three complex numbers $c_1 = a_1 + b_1 i$,$c_2 = a_2 + b_2 i$ and $c_3 = a_3 + b_3 i$ this function returns the area of the triangle formed by(if exists) the points c_1,c_2 and c_3.

- `get_perimeter double :(cdouble c1,cdouble c2,cdouble c3)`: Given three complex numbers $c_1 = a_1 + b_1 i$,$c_2 = a_2 + b_2 i$ and $c_3 = a_3 + b_3 i$ this function returns the perimeter of the triangle formed by(if exists) the points c_1,c_2 and c_3.

8 Example programs

8.1 Example program 1:

```
my_centroid cdouble : (cdouble c1,cdouble c2,cdouble c3)  {
    cdouble centroid;
    centroid = (c1+c2+c3)/3;
    return centroid;
}
main int :  {
    cint a(3,4);
    cint b(5,5),c(-101,100);
    cdouble centroid;
    centroid = my_centroid(a,b,c);
    choice(centroid eq get_centroid(a,b,c)) {
        cprint(centroid);
    }
    default {
        cprint(is_triangle(a,b,c));
    }
    return 0;
}
```

Figure 6: Output for table 'department' and k=10

8.2 Example program 2:

```
main int :  {
    cint a(3,4);
    cint b(5,5),c(-101,100);
    cdouble centroid;
    centriod = get_centroid(a,b,c);
    cprint(centroid);
    circumcente= get_circumcenter(a,b,c);
    cprint(circumcenter);
    orthocenter = get_orthocenter(a,b,c);
    cprint(orthocenter);
    choice (dist(centriod,circumcenter) eq dist(orthocenter,centroid)*2){
        cprint(1); //ratio verified
    }
    default {
        cprint(-1);
    }
    //circum centriod orthocenter
    //      2        1
    return 0;
}
```

Figure 7: Output for table 'department' and k=10