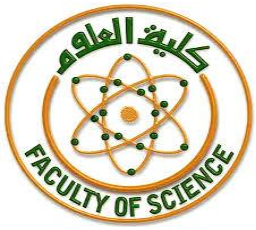


رقم البحث هو : 01704

عنوان المشروع :

Elimination of Common prefix in Compiler Design using C++

أسم المقرر : تصميم مؤلفات

رقم المقرر: COMP304

المستوى / الفرقة : الثالث

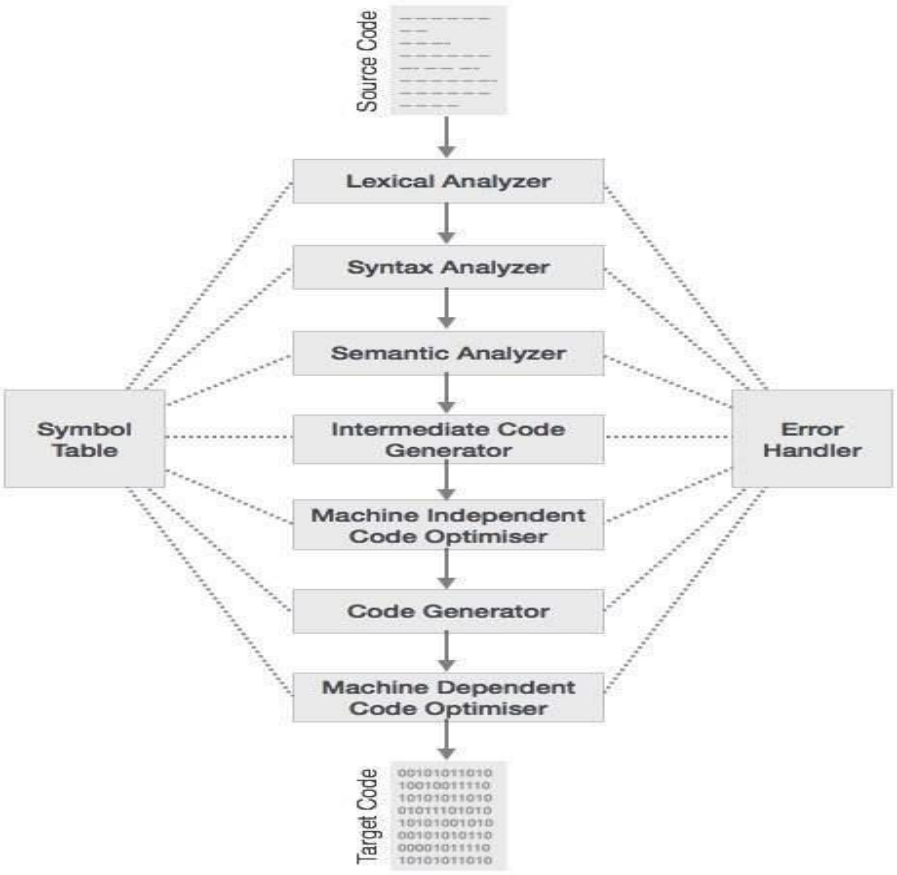
البرنامج/الشعبة : علوم حاسب

استاذ المقرر: د/اشرف مصطفي

Introduction:

As we know ,The compiler design includes a sequence of phases.

Each phase takes input from its previous stage, has its own representation of source program, and feeds its output to the next phase of the compiler.



And it was in its beginning Scanner (lexical analysis) . after that, we studied a different ways to build the Scanner , like : (Hand-Coded , regular expression with transaction diagram , regular expression with transaction table). And We took a method using a parser . And as we know parser get token from scanner an check ,

If this token the same token in the (grammar) which we call it (match) , If it's correct ,the parser get anther token from the scanner and so on.

So, the output is **an error** or **syntax tree**.

There is Two ways to parse input : (Top-down parsing , Bottom-up parsing)

Top-down parsing : we start with the start symbol and apply the production until You arrive at the desired string .

note: Top-down parsers cannot handle common prefix in a grammar , because the parser can not determine whether to expand a nonterminal

$$<A> \text{ to } a\beta_1 \text{ or } a\beta_2 \text{ or } \ldots.\text{or } a\beta_n$$

Before we start working on a program that works to remove common prefix , we will discuss first The theory that allows conversion the grammar with common prefix to a left factored grammar.

Eliminating Direct Common Prefix:

For all ,A $\in$ V , find the longest common prefix a that occurs in two or more right-hand sides of A if a != E then replace all of A productions,

$$<A> ::= a\beta_1 | a\beta_2 | \ldots. | a\beta_n | y$$

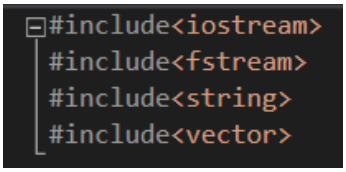
with : <A> ::=a<Z> |y

$$<Z> ::= \beta_1 | \beta_2 | \ldots. | \beta_n$$

Where Z is a new element of V.

Now , let's create a program to remove common prefix using c++ :

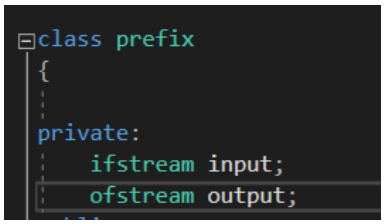
-I call (fstream , srting , vector) libraries to transact with files , Words , phrases and to save data.



-I use OOP and create a class (prefix) that contain a private data members which is

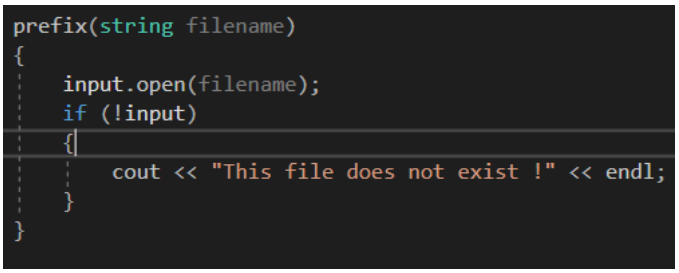
An object of (ifstream) to get data as input,

An object of (ofstream) to output the data



-And in the public there are constructor , destructor and function (check),

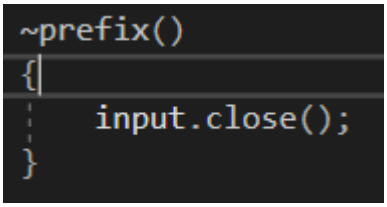
to process the grammar .



-In this constructor we get the name of file as a parameter and open it .

This (if statement) check if this file exist or not .

-This destructor used only to close the file .



Let's discuss about (check) function :

```
void check()
{
    output.open("outPut.txt");

    string s;
    int dash_no = 1;

    while (getline(input, s))
    {
```

- First we will open the output file to put data after process on it .
- Then I define a variable from type string to put data on it line by line.
- And (dash_on) it 's a variable I will explain it later .
- After that I create (while loop) which get the data line by line from input file.

We divided the line into two parts , A part before the (=) and the other after it to make my process on it.

```
int size = s.length();
int index_of_equal = s.find('=');

string before_equal = s.substr(0, index_of_equal + 1);
string after_equal = s.substr(index_of_equal + 1, size - (index_of_equal + 1));
```

-In this (if statement) we check if there is a (|) or not ,Because if there is no (|) so , this is a normal statement with no common prefix.

```
if (after_equal.find('|') == -1)
{
    output << s << "\n";
}
else
{
```

-I define a vector start with (1) to make it easy for me while tracing .

-This (for loop) put every the production rules in the vector (vec).

```
vector <string> vec(1);
string sub_string_from_after_equal;

for (int i = 0; i < after_equal.length(); i++)
{
    if (after_equal[i] != '|')
    {
        sub_string_from_after_equal += after_equal[i];
    }
    else
    {
        vec.push_back(sub_string_from_after_equal);
        sub_string_from_after_equal = "";
    }
}
vec.push_back(sub_string_from_after_equal);
```

-I put the longest common prefix in (prefix_string) by using this (while loop),

```
string prefix_string;
string compare_string = vec[1];
bool flag = true;
```

```
while (vec[2].find(compare_string) != 0)
{
    compare_string = compare_string.substr(0, compare_string.length() - 1);
}
```

-I get the prefix_string by compare the first two vector with each other ,but I put the first element of the vector in (compare_string) to make some change on it with out changing in the original element. And after that I put it in prefix_string.

-In this (if) I check if the length of the prefix_string == 0 so, it mean this statement is a normal statement with out common prefix.

So, the flag become false to output this statement put it as is.

```
if (prefix_string.length() == 0)
    flag = false;


if (!flag)
    output << s << "\n";
else
```

After that we will show how to output the data:

```
output << " *** " << before_equal << " " << prefix_string << " <x'" << dash_no << ">";
for (int i = 1; i < vec.size(); i++)
{
    if (vec[i].find(prefix_string) != 0)
        output << " | " << vec[i];
}
output<< "\n";
```

I've set (***) in the beginnnig of each stetment have common prefix for excellence and ease of vision , (dash_no) gradually increasing make each nonterminal different from the one before

```
output << "    <x'" << dash_no << ">";
output << " ::= ";
for (int i = 1; i < vec.size(); i++)
{
    if (vec[i].find(prefix_string) == 0)
    {
        if (vec[i].substr(prefix_string.length(), vec[i].length()).find(' ') == 0)
            output << "E";
        if (i > 1)
        {
            output << " | " << vec[i].substr(prefix_string.length(), vec[i].length());
        }
        else
        {
            output << vec[i].substr(prefix_string.length(), vec[i].length());
        }
    }
}
output << "\n";
dash_no++;
```

-Then this (for loop) makes the new line by putting the elements of the vector side by side

Note: if the element does not contain any thing except the (prefix_string)

So, I output Epsilon (‘E’).

Main function:

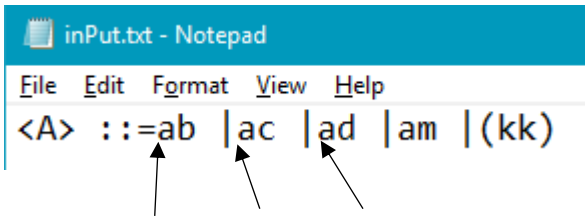
```
do {
    int choose;
    string name_of_file;

    cout << "press (1) : To run an example of (mini-java-BNF-grammer) and other exampels in output file." << endl;
    cout << "press (2) : To choose the name of the file you want!" << endl;
    cout << "press (3) : To exit!" << endl << endl;
    cin >> choose;

    if (choose == 1)
    {
        prefix x("inPut.txt");
        x.check();
        x.~prefix();
        break;
    }
```

- It contains a set of options that make the program easier as shown.

Now, let's solve some problems and its output in the file:

- I put some special cases while inserting the data in (input) file.

- Every line contain only one statement .
- In each statement we have to put the first two production rules with common prefix in the first ,then any production rule .
- After any (=) and (|) must start with no space.



Examples:

inPut.txt

```
<A> ::=ab |ac |ad |am
```

outPut.txt

```
*** <A> ::= a <x'1>
  <x'1> ::= b  | c  | d  | m
```

inPut.txt

```
<A> ::=ab |ac |ad |am |(kk)
```

outPut.txt

```
*** <A> ::= a <x'2> | (kk)
  <x'2> ::= b  | c  | d  | m
```

inPut.txt

```
<A> ::=ab |ac |(kk) |am |ad
```

outPut.txt

```
*** <A> ::= a <x'3> | (kk)
  <x'3> ::= b  | c  | m  | d
```

inPut.txt <Factor> ::=Identifier |Identifier[<ExprList>] |Identifier(<ExprList>)

outPut.txt
```
*** <Factor> ::= Identifier <x'4>
  <x'4> ::= E  | [<ExprList>]  | (<ExprList>)
```

inPut.txt <s> ::=If e Then s |If e then s Else s |a

outPut.txt
```
*** <s> ::= If e  <x'5> | a
  <x'5> ::= Then s  | then s Else s
```

→some normal statements :

inPut.txt
```
<program> ::=<main class> <class declarations>

<main class> ::=class <identifier> { public static void main ( ) { <statements> }   }
```

outPut.txt same in the outPut.txt

→normal statement with (|) but without common prefix:

inPut.txt <simple type> ::=int | boolean | void | < type identifier>

outPut.txt same in the outPut.txt

→Statements we put the common prefix in the begin:

inPut.txt
```
<statement> ::=if  ( <expr> ) <statement>  |if  ( <expr> ) <statement> else <statement>  |assert ( expr )
<expr> ::=<expr> <binary operator> <expr>  |<expr> [ <expr> ]  |<expr> . length  |new <simple type> [ <expr>
```

outPut.txt
```
*** <statement> ::= if  ( <expr> ) <statement>  <x'8> | assert ( expr )  | < local variable declaration>
  <x'8> ::= E  | else <statement>

*** <expr> ::= <expr>  <x'9> | new <simple type> [ <expr> ]  | new <type identifier> ( )  | ! <expr>
  <x'9> ::= <binary operator> <expr>  | [ <expr> ]  | . length
```

note: There is a supplement to those statements in the file (input.txt).

Conclusion:

By the end, we discussed how to build a program to solve how to eliminate the common prefix and we discussed the project detailed design in a simple report .

The source code and the (input.txt , output.txt) files with there data was created .

Sources:

(s.1) https://en.wikipedia.org/wiki/Parsing

(s.2) https://www.geeksforgeeks.org

(s.3) http://stackoverflow.com