11. recursive Descent parser

#include <stdio.h>

#include <unistd.h>

#include <string.h>

#include <ctype.h>

char input[10];

int i, error;

void E();

void T();

void Eprime();

void Tprime();

void F();

int main()

{

i = 0;

error = 0;

printf("Enter an arithmetic expression : "); // Eg: a+a\*a

gets(input);

E();

if (strlen(input) == i && error == 0)

printf("\nAccepted..!!!\n");

else

printf("\nRejected..!!!\n");

}

void E()

{

T();

Eprime();

}

void Eprime()

{

if (input[i] == '+')

{

i++;

T();

Eprime();

}

}

void T()

{

F();

Tprime();

}

void Tprime()

{

if (input[i] == '\*')

{

i++;

F();

Tprime();

}

}

void F()

{

if (isalnum(input[i]))

i++;

else if (input[i] == '(')

{

i++;

E();

if (input[i] == ')')

i++;

else

error = 1;

}

else

error = 1;

}

OUTPUT

Enter an arithmetic expression : a+a\*a

Accepted..!!!

12. shift reduce parser

// Including Libraries

#include <stdio.h>

#include<stdlib.h>

#include<string.h>

// Global Variables

int z = 0, i = 0, j = 0, c = 0;

// Modify array size to increase

// length of string to be parsed

char a[16], ac[20], stk[15], act[10];

// This Function will check whether

// the stack contain a production rule

// which is to be Reduce.

// Rules can be E->2E2 , E->3E3 , E->4

void check()

{

// Copying string to be printed as action

strcpy(ac, "REDUCE TO E -> ");

// c=length of input string

for (z = 0; z < c; z++)

{

// checking for producing rule E->4

if (stk[z] == '4')

{

printf("%s4", ac);

stk[z] = 'E';

stk[z + 1] = '\0';

// printing action

printf("\n$%s\t%s$\t", stk, a);

}

}

for (z = 0; z < c - 2; z++)

{

// checking for another production

if (stk[z] == '2' && stk[z + 1] == 'E' &&

stk[z + 2] == '2')

{

printf("%s2E2", ac);

stk[z] = 'E';

stk[z + 1] = '\0';

stk[z + 2] = '\0';

printf("\n$%s\t%s$\t", stk, a);

i = i - 2;

}

}

for (z = 0; z < c - 2; z++)

{

// checking for E->3E3

if (stk[z] == '3' && stk[z + 1] == 'E' &&

stk[z + 2] == '3')

{

printf("%s3E3", ac);

stk[z] = 'E';

stk[z + 1] = '\0';

stk[z + 2] = '\0';

printf("\n$%s\t%s$\t", stk, a);

i = i - 2;

}

}

return; // return to main

}

// Driver Function

int main()

{

printf("GRAMMAR is -\nE->2E2 \nE->3E3 \nE->4\n");

// a is input string

strcpy(a, "32423");

// strlen(a) will return the length of a to c

c = strlen(a);

// "SHIFT" is copied to act to be printed

strcpy(act, "SHIFT");

// This will print Labels (column name)

printf("\nstack \t input \t action");

// This will print the initial

// values of stack and input

printf("\n$\t%s$\t", a);

// This will Run upto length of input string

for (i = 0; j < c; i++, j++)

{

// Printing action

printf("%s", act);

// Pushing into stack

stk[i] = a[j];

stk[i + 1] = '\0';

// Moving the pointer

a[j] = ' ';

// Printing action

printf("\n$%s\t%s$\t", stk, a);

// Call check function ..which will

// check the stack whether its contain

// any production or not

check();

}

// Rechecking last time if contain

// any valid production then it will

// replace otherwise invalid

check();

// if top of the stack is E(starting symbol)

// then it will accept the input

if (stk[0] == 'E' && stk[1] == '\0')

printf("Accept\n");

else // else reject

printf("Reject\n");

}

OUTPUT

GRAMMAR is -

E->2E2

E->3E3

E->4

stack input action

$ 32423$ SHIFT

$3 2423$ SHIFT

$32 423$ SHIFT

$324 23$ REDUCE TO E -> 4

$32E 23$ SHIFT

$32E2 3$ REDUCE TO E -> 2E2

$3E 3$ SHIFT

$3E3 $ REDUCE TO E -> 3E3

$E $ Accept

13. constant propagation

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#include<stdlib.h>

void input();

void output();

void change(int p, char \*res);

void constant();

struct expr

{

char op[2], op1[5], op2[5], res[5],x[5];

int ag;

} arr[10];

int n;

int main()

{

input();

constant();

output();

}

void input()

{

int i;

printf("\n\nEnter the maximum number of expressions : ");

scanf("%d", &n);

printf("\nEnter the input : \n");

printf("\n{for assigning value eg. (variable)a=(value)3=-}");

for (i = 0; i < n; i++)

{

scanf("%s", arr[i].res);

scanf("%s", arr[i].x);

scanf("%s", arr[i].op1);

scanf("%s", arr[i].op);

scanf("%s", arr[i].op2);

arr[i].ag = 0;

}

}

void constant()

{

int i;

int op1, op2, res;

char op, res1[5];

for (i = 0; i < n; i++)

{

if (isdigit(arr[i].op1[0]) && isdigit(arr[i].op2[0]) || strcmp(arr[i].op, "=") == 0)

{

op1 = atoi(arr[i].op1);

op2 = atoi(arr[i].op2);

op = arr[i].op[0];

switch (op)

{

case '+':

res = op1 + op2;

break;

case '-':

res = op1 - op2;

break;

case '\*':

res = op1 \* op2;

break;

case '/':

res = op1 / op2;

break;

case '=':

res = op1;

break;

}

sprintf(res1, "%d", res);

arr[i].ag = 1; /\*eliminate expr and replace any operand below that uses

result of this expr \*/

change(i, res1);

}

}

}

void output()

{

int i = 0;

printf("\nOptimized code is : ");

for (i = 0; i < n; i++)

{

if (!arr[i].ag)

{

printf("\n%s = %s %s %s", arr[i].res, arr[i].op1, arr[i].op, arr[i].op2);

}

}

}

void change(int p, char \*res)

{

int i;

for (i = p + 1; i < n; i++)

{

if (strcmp(arr[p].res, arr[i].op1) == 0)

strcpy(arr[i].op1, res);

else if (strcmp(arr[p].res, arr[i].op2) == 0)

strcpy(arr[i].op2, res);

}

}

OUTPUT

Enter the maximum number of expressions : {for assigning value eg. a=3--4

{for assigning value eg. (variable)a=(value)3=-}

Enter the input :

a = 3 = -

t1 = a + b

t2 = a + c

t3 = t1 + t2

Optimized code is :

t1 = 3 + b

t2 = 3 + c

t3 = t1 + t2

14. Intermediate code generation

#include <stdio.h>

#include <string.h>

int i = 1, j = 0, no = 0, tmpch = 90;

char str[100], left[15], right[15];

void ndopr();

void explore();

void eft(int);

void fright(int);

struct exp

{

int pos;

char op;

} k[15];

int main()

{

printf("\t\tINTERMEDIATE CODE GENERATION\n\n");

printf("Enter the Expression :");

scanf("%s", str);

printf("The intermediate code:\n");

ndopr();

explore();

}

void ndopr()

{

for (i = 0; str[i] != '\0'; i++)

if (str[i] == ':')

{

k[j].pos = i;

k[j++].op = ':';

}

for (i = 0; str[i] != '\0'; i++)

if (str[i] == '/')

{

k[j].pos = i;

k[j++].op = '/';

}

for (i = 0; str[i] != '\0'; i++)

if (str[i] == '\*')

{

k[j].pos = i;

k[j++].op = '\*';

}

for (i = 0; str[i] != '\0'; i++)

if (str[i] == '+')

{

k[j].pos = i;

k[j++].op = '+';

}

for (i = 0; str[i] != '\0'; i++)

if (str[i] == '-')

{

k[j].pos = i;

k[j++].op = '-';

}

}

void explore()

{

i = 1;

while (k[i].op != '\0')

{

eft(k[i].pos);

fright(k[i].pos);

str[k[i].pos] = tmpch--;

printf("\t%c := %s%c%s\t\t", str[k[i].pos], left, k[i].op, right);

printf("\n");

i++;

}

fright(-1);

if (no == 0)

{

eft(strlen(str));

printf("\t%s := %s", right, left);

}

printf("\t%s := %c", right, str[k[--i].pos]);

}

void eft(int x)

{

int w = 0, ag = 0;

x--;

while (x != -1 && str[x] != '+' && str[x] != '\*' && str[x] != '=' && str[x] != '\0' &&

str[x] != '-' && str[x] != '/' && str[x] != ':')

{

if (str[x] != '$' && ag == 0)

{

left[w++] = str[x];

left[w] = '\0';

str[x] = '$';

ag = 1;

}

x--;

}

}

void fright(int x)

{

int w = 0, ag = 0;

x++;

while (x != -1 && str[x] != '+' && str[x] != '\*' && str[x] != '\0' && str[x] != '=' &&

str[x] != ':' && str[x] != '-' && str[x] != '/')

{

if (str[x] != '$' && ag == 0)

{

right[w++] = str[x];

right[w] = '\0';

str[x] = '$';

ag = 1;

}

x++;

}

}

OUTPUT

INTERMEDIATE CODE GENERATION

Enter the Expression :w:=a\*b+c/d-e/f+g\*h

The intermediate code:

Z := c/d

Y := e/f

X := a\*b

W := g\*h

V := X+Z

U := Y+W

T := V-U

w := T