UEM

श्रद्धावान लभते ज्ञानम्

UNIVERSITY OF ENGINEERING & MANAGEMENT

Good Education, Good Jobs

University of Engineering & Management, Kolkata

Department of Computer Science & Engineering

Course: B.Tech. CSE / CSE (AIML) / CSE (IOT-CYS-BCT) / CSBS

Semester: 6th

Paper Name: Compiler Design Laboratory

Paper Code: PCC-CSE691

Week 1

Programs on the following topic:

Develop a lexical analyzer to recognize a few patterns in C. (Ex. identifiers, constants, Comments, operators etc.)

1. Write a C program to check if a user given string is a valid identifier or not?

```
#include <stdio.h>
#include <string.h>

int isValid(char *str, int n)
{
	if (!((str[0] >= 'a' && str[0] <= 'z')
		|| (str[0] >= 'A' && str[0] <= 'Z')
		|| str[0] == '_'))
		return 0;

	for (int i = 1; i < n; i++) {
		if (!((str[i] >= 'a' && str[i] <= 'z')
			|| (str[i] >= 'A' && str[i] <= 'Z')
			|| (str[i] >= '0' && str[i] <= '9')
			|| str[i] == '_'))
			return 0;
	}

	return 1;
}

int main()
{
	char str[100];
	printf("Enter an identifier: ");
	scanf("%s",str);
	int n = strlen(str);
	if(isValid(str,n))
		printf("Valid\n");
	else
		printf("Invalid\n");

	return 0;
}
```

2. Write a C program to check if a user given C program statement is a valid Comment or not?

```
#include <stdio.h>
#include <string.h>

int isComment(char *line, int n)
{
	if ((line[0] == '/' && line[1] == '/'
		&& line[2] != '/') || (line[n - 2] == '*'
		&& line[n - 1] == '/' && line[0] == '/' && line[1] == '*'))
	return 1;

	else
	return 0;
}

int main()
{
	char str[100];
	printf("Enter an identifier: ");
	scanf("%s",str);
	int n = strlen(str);
```

```
    if(isComment(str,n))
        printf("It is a comment\n");
    else
        printf("It is not a comment\n");

    return 0;
}
```

3. Write a C program to read a program written in a file and remove all comments. After removing all comments, rewrite the program in a separate file.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    FILE *fp1=fopen("input.c","r"),*fp2=fopen("output.c","w");
    char ch;
    if(fp1==NULL || fp2==NULL)
        printf("Error while opening a file for %s",(fp1==NULL)?"reading":"writing");

    else
    {
        while((ch=fgetc(fp1))!=EOF)
        {
            if(ch=='"')
            {
                fputc(ch,fp2);
                while((ch=fgetc(fp1))!=EOF)
                    {

                        if(ch!='"')
                        fputc(ch,fp2);
                        else
                        break;
                    }
            }
            else if(ch=='/')
            {
                if(((ch=fgetc(fp1))!=EOF)&& ch=='/')
                {
                    while((ch=fgetc(fp1))!=EOF)
                    {
                        if(ch!='\n')
                        continue;
                        else
                        {
                            fputc('\n',fp2);
                            break;
                        }

                    }
                    continue;
                }
                else if(ch=='*')
                {
                    while((ch=fgetc(fp1))!=EOF)
                    {
                        if(ch!='*')
                        continue;
                        else if(((ch=fgetc(fp1))!=EOF)&& ch=='/')
                            break;
                    }
                    continue;
                }
                fputc('/',fp2);
            }
```

```
                  fputc(ch,fp2);
            }
            fclose(fp1);
            fclose(fp2);
            fp1=fopen("outputrc.c","r");
            while((ch=fgetc(fp1))!=EOF)
            {
                  printf("%c",ch);
            }
            fclose(fp1);
      }
      return 0;
}
```

4. Write a C program to convert an infix statement into a postfix statement.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAX_EXPR_SIZE 100

int precedence(char operator) {
      switch (operator) {
            case '+':
            case '-':
                  return 1;
            case '*':
            case '/':
                  return 2;
            case '^':
                  return 3;
            default:
                  return -1;
      }
}

int isOperator(char ch) {
      return (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '^');
}

char *infixToPostfix(char *infix) {
      int i, j;
      int len = strlen(infix);
      char *postfix = (char *)malloc(sizeof(char) * (len + 2));
      char stack[MAX_EXPR_SIZE];
      int top = -1;

      for (i = 0, j = 0; i < len; i++) {
            if (infix[i] == ' ' || infix[i] == '\t')
                  continue;
            if (isalnum(infix[i])) {
                  postfix[j++] = infix[i];
            } else if (infix[i] == '(') {
                  stack[++top] = infix[i];
            } else if (infix[i] == ')') {
                  while (top > -1 && stack[top] != '(')
                        postfix[j++] = stack[top--];
                  if (top > -1 && stack[top] != '(')
                        return "Invalid Expression";
                  else
                        top--;
            } else if (isOperator(infix[i])) {
                  while (top > -1 && precedence(stack[top]) >= precedence(infix[i]))
                        postfix[j++] = stack[top--];
                  stack[++top] = infix[i];
            }
      }

      while (top > -1)
            postfix[j++] = stack[top--];

      postfix[j] = '\0';
      return postfix;
}

int main() {
      char infix[MAX_EXPR_SIZE];
      printf("Enter an infix expression: ");
      gets(infix);
      char *postfix = infixToPostfix(infix);
      printf("Postfix expression: %s\n", postfix);
      free(postfix);
      return 0;
}
```

5. Write a C program to evaluate an arithmetic expression which is given as a string. Consider the input has no parentheses and contains the following operators only: +, -, *, /

```
#include<stdio.h>
int main()
{
        int a,b,c,d,f,g,h,p,q;
        printf("Enter values for the expression a+b*c-d/f+g-h*p/q: ");
        scanf("%d+%d*%d-%d/%d+%d-%d*%d/%d",&a,&b,&c,&d,&f,&g,&h,&p,&q);
        printf("Value= %d",a+b*c-d/f+g-h*p/q);
}
```

Week 2

Programs on the following topic:
Implementation of Lexical Analyzer using Lex Tool

6. Write a Lex Program to count the number of vowels and consonants in a given string.

```
%{
int vow, con, printf(const char*, ...);
%}

%%
[aeiou] {printf("Current vowel = %s\n", yytext);
       vow++;}
[bcdfghjklmnpqrstvwxyz] {printf("Current consonant = %s\n", yytext);
                    con++;}
\n return 0;
%%

int yywrap(void){}
int main (void)
{
       yylex();
       printf("vowels = %d, consonants = %d\n", vow,con);
}
```

7. Write a Lex Program to count the number of characters, words, spaces, end of lines in a given input file.

```
%{
#include<stdio.h>
int lc=0,sc=0,tc=0,ch=0,wc=0;
%}

%%
[\n] { lc++; ch+=yyleng;}
[  \t] { sc++; ch+=yyleng;}
[^\t] { tc++; ch+=yyleng;}
[^\t\n ]+ { wc++;  ch+=yyleng;}
%%

int yywrap(void){}
int main (void)
{
       yyin = fopen("sample.txt","r");
       //printf("Enter the Sentence: ");
       yylex();
       printf("Number of Lines = %d\n",lc);
       printf("Number of Spaces = %d\n",sc);
       printf("Number of Tabs = %d\n",tc);
       printf("Number of Words = %d\n",wc);
       printf("Number of Characters = %d\n",ch);
       return 0;
}
```

8. Write a Lex Program to count no of: a) +ve and –ve integers b) +ve and –ve fractions.

```
%{
#include<stdio.h>
int posint=0,negint=0,posfraction=0,negfraction=0;
%}

%%
[+]?[0-9]+ { posint++;}
[-][0-9]+ { negint++;}
[+]?[0-9]*\.[0-9]+ { posfraction++;}
[-][0-9]*\.[0-9]+ { negfraction++;}
```

```
%%

int yywrap(void){}
int main (void)
{
      //yyin = fopen("sample.txt","r");
      printf("Enter the numbers: ");
      yylex();
      printf("Number of Positive Integer = %d\n",posint);
      printf("Number of Negative Integer = %d\n",negint);
      printf("Number of Positive Fraction = %d\n",posfraction);
      printf("Number of Negative Fraction = %d\n",negfraction);
      return 0;
}
```

9. Write a Lex Program to count the no of comment line in a given C program. Also eliminate them and copy that program into separate file.

```
%{
#include<stdio.h>
int nc1=0,nc=0;
%}

%%
"/*"[a-zA-Z0-9\n\t ]*"*/"  {nc1++;}
"//"[a-zA-Z0-9\t ]*"\n"   {nc++;}
%%

int main()
{
      yyin=fopen("sample.c","r");
      yyout=fopen("output.c","w");
      yylex();
      printf("The number of Singleline comments = %d\n",nc);
      printf("The number of Multiline comments = %d\n",nc1);
      fclose(yyin);
      fclose(yyout);
}

int yywrap( )
{
      return 1;
}
```

10. Write a Lex Program to count the no of ‘scanf’ and ‘printf’ statements in a C program. Replace them with ‘readf’ and ‘writef’ statements respectively

```
%{
#include<stdio.h>
int pf=0,sf=0;
%}

%%
"printf" { fprintf(yyout,"writef"); pf++;}
"scanf" { fprintf(yyout,"readf"); sf++;}
%%

int main()
{
      yyin=fopen("sample.c","r");
      yyout=fopen("output.c","w");
      yylex();
      printf("The number of printf statements  = %d\n",pf);
      printf("The number of scanf statements  = %d\n",sf);
      fclose(yyin);
      fclose(yyout);
}

int yywrap( )
{
        return 1;
}
```

11. Write a Lex Program to recognize a valid arithmetic expression and identify the identifiers and operators present. Print them separately.

```
%{
#include <stdio.h>
#include <string.h>
        int operators_count = 0, operands_count = 0, valid = 1, top = -1, l = 0, j = 0;
        char operands[10][10], operators[10][10], stack[100];
%}
%%
"(" {top++;stack[top] = '(';}
"{" {top++;stack[top] = '{';}
"[" {top++;stack[top] = '[';}
")" {
        if (stack[top] != '(') {
                valid = 0;
        }
        else if(operands_count>0 && (operands_count-operators_count)!=1){
                valid=0;
        }
        else{
                top--;
                operands_count=1;
                operators_count=0;
        }
}
"}" {
        if (stack[top] != '{') {
                valid = 0;
        }
        else if(operands_count>0 && (operands_count-operators_count)!=1){
                valid=0;
        }
        else{
                top--;
                operands_count=1;
                operators_count=0;
        }
}
"]" {
        if (stack[top] != '[') {
                valid = 0;
        }
        else if(operands_count>0 && (operands_count-operators_count)!=1){
                valid=0;
        }
        else{
                top--;
                operands_count=1;
                operators_count=0;
        }

}
"+"|"-"|"*"|"/" {operators_count++;strcpy(operators[l], yytext);l++;}
[0-9]+|[a-zA-Z][a-zA-Z0-9_]* {operands_count++;strcpy(operands[j], yytext);j++;}
%%


int yywrap()
```

```
{
      return 1;
}
int main()
{
      int k;
      printf("Enter the arithmetic expression: ");
      yylex();

      if (valid == 1 && top == -1) {
            printf("\nValid Expression\n");
            printf("Operators count: %d\n",operators_count);
            printf("Operands count: %d\n",operands_count);
      }
      else
            printf("\nInvalid Expression\n");

      return 0;
}
```

12. Write a Lex Program to recognize whether a given sentence is simple or compound.

```
%{
      #include<stdio.h>
      int flag=0;
%}

%%
and |
or |
but |
because |
if |
then |
nevertheless  { flag=1; }
.  ;
\n  { return 0; }
%%

int main()
{
      printf("Enter the sentence:\n");
      yylex();
      if(flag==0)
            printf("Simple sentence\n");
      else
            printf("compound sentence\n");
}

int yywrap( )
{
      return 1;
}
```

13. Write a Lex Program to implement arithmetic calculator.

```
%{
int op = 0,i;
float a, b;
%}

dig [0-9]+|([0-9]*)"."([0-9]+)
add "+"
sub "-"
mul "*"
div "/"
```

```
pow "^"
ln \n

%%
{dig} {digi();}
{add} {op=1;}
{sub} {op=2;}
{mul} {op=3;}
{div} {op=4;}
{pow} {op=5;}
{ln} {printf("The Answer :%f\n\n=================================\n\nEnter the calculation
:\n",a);}
%%

digi()
{
    if(op==0)
        a=atof(yytext);

    else
    {
        b=atof(yytext);

        switch(op)
        {
            case 1: a=a+b;
                    break;

            case 2: a=a-b;
                    break;

            case 3: a=a*b;
                    break;

            case 4: a=a/b;
                    break;

            case 5: for(i=a;b>1;b--)
                        a=a*i;
                    break;
        }
        op=0;
    }
}

int main(int argv,char *argc[])
{
    printf("\n\n=================================\n\nEnter the calculation :\n");
    yylex();
}

int yywrap()
{
    return 1;
}
```

Week 3

Programs on the following topic:
Generate YACC specification for a few syntactic categories.

14. Write a Lex Program to recognize and count the number of identifiers in a given input file.

Lex Part

```
%{
#include<stdio.h>
int count=0;
char ch=0;
%}

digit[0-9]
letter[a-zA-Z_]

%%
{letter}({letter}|{digit})* {count++;}
. ;
\n ;
%%
int yywrap(void){}
int main()
{
	yyin=fopen("sample.c","r");
	yylex();
	printf("count: %d\n",count);
	fclose(yyin);
	return 0;
}
```

15. Write a YAAC Program to test the validity of a simple expression involving operators +, -, * and /

Lex Part

```
%{
#include "y.tab.h"
extern yylval;
%}

%%
[0-9]+ {yylval = atoi(yytext);
	return NUMBER;}

[a-zA-Z]+ { return ID; }
[ \t]+ ;

\n { return 0; }
. { return yytext[0]; }

%%
int yywrap(void){}
```

YACC Part

```
%{
#include <stdio.h>
#include <stdlib.h>
%}

%token NUMBER ID
```

```
%left '+' '-'
%left '*' '/'

%%

T :
       T '+' T
       | T '-' T
       | T '*' T
       | T '/' T
       | '-' NUMBER
       | '-' ID
       | '(' T ')'
       | NUMBER
       | ID ;
%%

int main() {
       printf("Enter the expression\n");
       yyparse();
       printf("\nExpression is valid\n");
}

int yyerror(char* s) {
       printf("\nExpression is invalid\n");
       exit(0);
}
```

16. Write a YAAC Program to recognize nested IF control statements and display the levels of nesting.

Lex Part

```
%{
#include "y.tab.h"
%}
%%
if return IF;
[{] return BEGIN1;
[}] return END1;
%%
```

YACC Part

```
%{
# include <stdio.h>
# include <stdlib.h>
int counter, yylex(void), yyerror(const char *);
%}
%token IF BEGIN1 END1
%%
S : I  {printf("nesting = %d\n",counter);}
I : IF BEGIN1 I END1 {counter++;}
  | BEGIN1 END1 {;}
  | {;}
%%
int main(void)
{
       yyparse();
}
int yyerror(const char* s) {
       printf("\n%s\n",s);
       exit(1);
}
```

17. Write a YAAC Program to check the syntax of a simple expression involving operators +, -, * and /

Lex Part

```
%{
#include"17.tab.h"
%}

%%
[0-9]+ {return NUMBER;}
[a-zA-Z][a-zA-Z0-9_]* {return ID;}
\n {return NL;}
. {return yytext[0];}
%%
```

YACC Part

```
%{
#include<stdio.h>
#include<stdlib.h>
%}

%token NUMBER ID NL
%left '+' '-'
%left '*' '/'

%%
stmt: exp NL {printf("valid expression\n"); exit(0);}
;
exp: exp '+' exp | exp '-' exp | exp '*' exp | exp '/' exp | '(' exp ')' | ID | NUMBER
;
%%

int yyerror(char *msg)
{
 printf("Invalid expression\n");
 exit(0);
}

main()
{
 printf("enter the expression: \n");
 yyparse();
}
```

18. Write a YAAC Program to evaluate an arithmetic expression involving operating +, -, * and /

Lex Part

```
%{
#include "18.tab.h"
extern int yylval;
%}

%%
[0-9]+ { yylval = atoi(yytext);
       return NUMBER;
       }

[a-zA-Z]+ { return ID; }
[ \t]+        ;

\n            { return 0; }
.             { return yytext[0]; }
%%
```

YACC Part

```
%{
#include <stdio.h>
# include <stdlib.h>
int yylex(void), yyerror(const char *);
%}

%token NUMBER ID
%left '+' '-'
%left '*' '/'


%%
E : T   {printf("Result = %d\n", $$); return 0;}

T :
       T '+' T { $$ = $1 + $3; }
       | T '-' T { $$ = $1 - $3; }
       | T '*' T { $$ = $1 * $3; }
       | T '/' T { $$ = $1 / $3; }
       | '-' NUMBER { $$ = -$2; }
       | '-' ID { $$ = -$2; }
       | '(' T ')' { $$ = $2; }
       | NUMBER { $$ = $1; }
       | ID { $$ = $1; };
%%

int main() {
       printf("Enter the expression\n");
       yyparse();
}

int yyerror(const char* s) {
       printf("\nExpression is invalid\n");
}
```

19. Write a YAAC Program to recognize a valid variable, which starts with a letter, followed by any number of letters or digits.

Lex Part

```
%{
    #include "y.tab.h"
%}

%%
[a-zA-Z_][a-zA-Z_0-9]* return letter;
[0-9]                  return digit;
.                      return yytext[0];
\n                     return 0;
%%

int yywrap()
{
return 1;
}
```

YACC Part

```
%{

#include<stdio.h>
int yylex(void), yyerror(const char *);
int valid=1;

%}

%token digit letter

%%

start : letter s

s :     letter s

      | digit s

      |

      ;

%%

int yyerror(const char* s)

{
    printf("\nIts not a identifier!\n");
    valid=0;
    return 0;
}

int main()

{
    printf("\nEnter a name to tested for identifier ");
    yyparse();
    if(valid)
    {
        printf("\nIt is a identifier!\n");

    }
}
```

20. Write a YAAC Program to recognize strings ‘aaab’, ‘abbb’, ‘ab’ and ‘a’ using grammar (a^n b^n , n>=0)

Lex Part

```
%{
  #include "20.tab.h"
%}

%%
.   return yytext[0];
\n  return 0;
%%

int yywrap()
 {
  return 1;
 }
```

YACC Part

```
%{
#include<stdio.h>
#include<stdlib.h>
int yylex(void), yyerror(const char *);
int valid=1;
%}
```

```
%token 'a' 'b'
%%
S :    'a' S 'b' | ;
%%

int yyerror(const char *msg)
{
    printf("invalid string\n");
    valid=0;
    return 0;
}

int main()
{
    printf("enter the string\n");
    yyparse();
    if(valid)
        printf("valid string\n");
}
```

21. Write a YAAC Program to recognize the grammar (an b, n>=10)

Lex Part

```
%{
  #include "21.tab.h"
%}

%%
.   return yytext[0];
\n  return 0;
%%

int yywrap()
 {
  return 1;
 }
```

YACC Part

```
%{
#include<stdio.h>
#include<stdlib.h>
int yylex(void), yyerror(const char *);
int valid=1;
%}
%token 'a' 'b'
%%
S : 'a' 'a' 'a' 'a' 'a' 'a' 'a' 'a' 'a' A 'b' ;
A : 'a' A |  ;
%%

int yyerror(const char *msg)
{
    printf("invalid string\n");
    valid=0;
    return 0;
}

int main()
{
    printf("enter the string\n");
    yyparse();
    if(valid)
        printf("valid string\n");
}
```

22. Write a YACC Program to implement arithmetic calculator.

Lex Part

```
%{
#include "22.tab.h"
int atoi(const char *);
%}
digit [0-9]
%%
{digit}+ {
             yylval = atoi(yytext);
             return NUM;
       }
. return *yytext;
\n yyterminate();
%%
int yywrap()
 {
  return 1;
 }
```

YACC Part

```
%{
#include <math.h>
#include<stdio.h>
int yylex(void), yyerror(const char *);
int flag=0;
%}

%token NUM
%left '+' '-'
%left '*' '/' '%'
%left '^'
%left '(' ')'

%%

S: E    {   printf("\nResult=%d\n", $$);
            return 0;
        };

E:  E'+'E {$$=$1+$3;}
    |E'-'E {$$=$1-$3;}
    |E'*'E {$$=$1*$3;}
    |E'/'E {$$=$1/$3;}
    |E'%'E {$$=$1%$3;}
    |E'^'E {$$=(int)pow($1,$3);}
    |'('E')' {$$=$2;}
    | NUM {$$=$1;}
 ;
%%
int yyerror(const char *e)
{
   printf("\nEntered arithmetic expression is Invalid\n\n");
   flag=1;
}
int main()
{
   printf("\nEnter Any Arithmetic Expression :\n");
   yyparse();
   if(flag==0)
   printf("\nEntered arithmetic expression is Valid\n\n");
}
```

Week 4

Programs on the following topic: Implementation of Symbol Table

23. Write a Program to implement Symbol Table.

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
void main()
{
    int i = 0, j = 0, x = 0, n;
    void *p, *add[10000];
    char ch, srch, b[15000], d[15000], c;
    printf("Expression terminated by $:");
    while ((c = getchar()) != '$')
    {
        b[i] = c;
        i++;
    }
    n = i - 1;
    printf("Given Expression:");
    i = 0;
    while (i <= n)
    {
        printf("%c", b[i]);
        i++;
    }
    printf("\n Symbol Table\n");
    printf("Symbol \t addr \t\t type");
    while (j <= n)
    {
        c = b[j];
        if (isalpha(toascii(c)))
        {
            p = malloc(c);
            add[x] = p;
            d[x] = c;
            printf("\n%c \t %d \t identifier\n", c, p);
            x++;
            j++;
        }
        else
        {
            ch = c;
            if (ch == '+' || ch == '-' || ch == '*' || ch == '=')
            {
                p = malloc(ch);
                add[x] = p;
                d[x] = ch;
                printf("\n %c \t %d \t operator\n", ch, p);
                x++;
                j++;
            }
        }
    }
}
```

Week 6

Programs on the following topic: Implement type checking

25. Write a C program to implement type checking.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int n, i, k, flag = 0;
    char vari[1500], typ[1500], b[1500], c;
    printf("Enter the number of variables:");
    scanf(" %d", &n);
    for (i = 0; i < n; i++)
    {
        printf("Enter the variable[%d]:", i);
        scanf(" %c", &vari[i]);
        printf("Enter the variable-type[%d](float-f,int-i):", i);
        scanf(" %c", &typ[i]);
        if (typ[i] == 'f')
            flag = 1;
    }
    printf("Enter the Expression(end with $):");
    i = 0;
    getchar();
    while ((c = getchar()) != '$')
    {
        b[i] = c;
        i++;
    }
    k = i;
    for (i = 0; i < k; i++)
    {
        if (b[i] == '/')
        {
            flag = 1;
            break;
        }
    }
    for (i = 0; i < n; i++)
    {
        if (b[0] == vari[i])
        {
            if (flag == 1)
            {
                if (typ[i] == 'f')
                {
                    printf("\nthe datatype is correctly defined..!\n");
                    break;
                }
                else
                {
                    printf("Identifier %c must be a float type..!\n", vari[i]);
                    break;
                }
            }
            else
            {
                printf("\nthe datatype is correctly defined..!\n");
                break;
            }
        }
    }
    return 0;
}
```

Week 9

Programs on the following topic: Construction of DAG

28. Write a C program to implement DAG.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MIN_PER_RANK 1
#define MAX_PER_RANK 5
#define MIN_RANKS 3
#define MAX_RANKS 5
#define PERCENT 30
void main()
{
	int i, j, k, nodes = 0;
	srand(time(NULL));
	int ranks = MIN_RANKS + (rand() % (MAX_RANKS - MIN_RANKS + 1));
	printf("DIRECTED ACYCLIC GRAPH\n");
	for (i = 1; i < ranks; i++)
	{
		int new_nodes = MIN_PER_RANK + (rand() % (MAX_PER_RANK - MIN_PER_RANK + 1));
		for (j = 0; j < nodes; j++)
			for (k = 0; k < new_nodes; k++)
				if ((rand() % 100) < PERCENT)
					printf("%d->%d;\n", j, k + nodes);
		nodes += new_nodes;
	}
}
```