“Intermediate Code Generation for sub-C language”

Submitted in fulfillment of the requirements of the degree of

(Bachelor’s of Technology)

by

Akash Singh

Roll- 157205

Reg- 961517

Faculty:

Professor T Ramesh

Department of Computer Science and Engineering

National Institute of Technology, Warangal

Acknowledgement

The project would not have been possible without the guidance of Prof. T Ramesh. The knowledge he imparted to us in this subject immensely helped in getting an in depth understanding all the complexities involved in the design of the part of the compiler constructed in the Project and thereafter solving them.

I would also like to thank my friends, discussing with whom clarified many edge cases and highlighted certain areas to be taken special care of.

Lastly, I would like to thank my parents for their support which inspires me to work harder everyday.

Table of Contents

Project Description

This assignment involves the construction of an intermediate code generator for a sub-C language. The input to the application is a sub-C code and the output is the three address intermediate code for the same.

A sub-C language is a subset of the C language supporting the following features:

1. **ASSIGNMENT**(=,+=,-=,*=,/=)
2. **BINARY OPERATOR**
   (a) ADDITION(+)
   (b) SUBTRACTION(-)
   (c) MULTIPLICATION(*)
   (d) DIVISION(/)
   (e) EXPONENTIATION(@)
3. **BITWIZE OPERATOR**
   (a) LOGICAL OR(|)
   (b) LOGICAL AND(&)
   (c) NEGATION(~)
   (d) XOR(^)
4. **LOGICAL OPERATOR**
   (a) AND(&&)
   (b) OR(||)
5. **RELATIONAL OPERATOR**
   (a) EQUALS(==)
   (b) NOT EQUALS(!=)
   (c) LESS THAN(<)
   (d) LESS THAN OR EQUAL TO(<=)
   (e) GREATER THAN(>)
   (f) GREATER THAN OR EQUAL TO(>=)
6. **ASSIGNMENT STATEMENTS**
7. **ITERATIVE STATEMENTS**(ONLY while)
8. **CONDITIONAL STATEMENTS**
   (a) IF
   (b) IF ELSE
   (c) SWITCH
9. **IDENTIFIERS**
10. **FUNCTIONS**

The application is constructed using 4 files whose details and metrics are listed as follows:

1. **LEX SPECIFICATION-495 lines-**Identifies the tokens required for identifying different constructs.
2. **YACC SPECIFICATION-760 lines-**Defines the different grammar constructs for the sub C language and also generated the nested structure for the intermediate code generator
3. **HEADER.H-89 lines-**Defines various helper functions used by the program
4. **CLASSES.H-353 lines-**Defines the classes for different constructs used in the inetrmediate code generation

Project Decomposition

The stated problem statement decomposes into following subcategories:

1. **Tokenizing**:This is done by the lex specification file.

2. **Parsing**:This identifies the proper syntax of the grammar for our sub-C language. This is done in the YACC Specification.

3. **Generating the intermediate Code**:This generates the final intermediate code and is interleaved with the parsing. This is also done in the YACC specification.

Flow Diagram

Figure 2 shows the flow diagram of the parser construction.

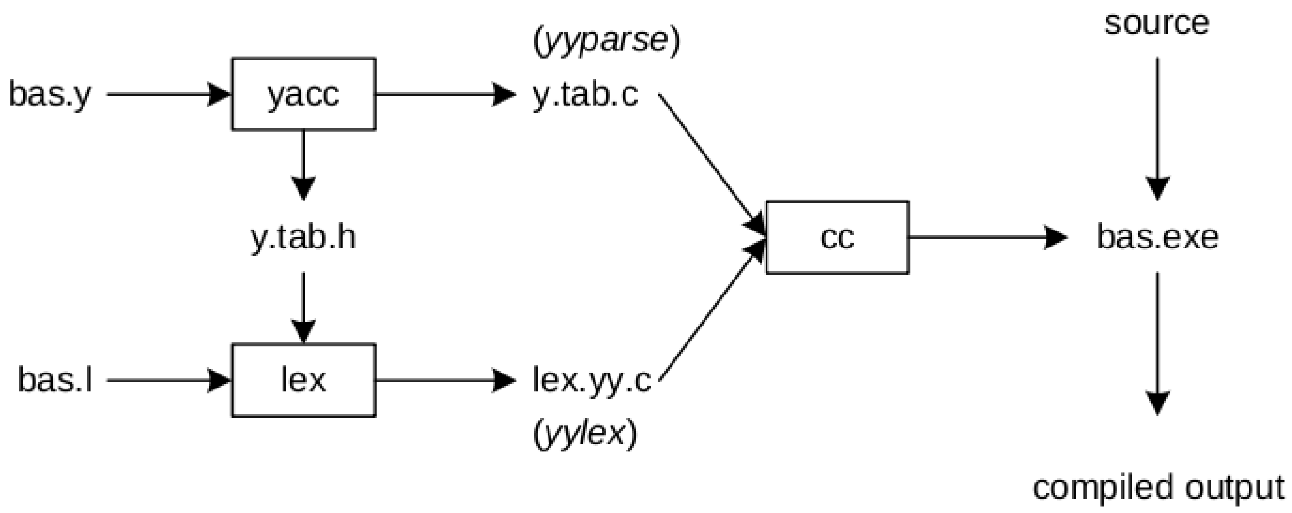


Figure 2: Building a Compiler with Lex/Yacc

Grammar Design

The design of the grammar follows a systematic definition.
In this section the relation and definition of each non terminal symbol in the grammar has been elaborated.

What each non-terminal derives has also been shown at required places in parenthesis and bold letters. Words in bold letters and in uppercase imply tokens generated by the lexical analyzer. The yylval of yacc carries the required values for the token.

The general explanation also explains the expected program structure of the sub-C language programs under consideration.

- **Program:** The program is defined to be composed of one or more succeeding components(**components**).

- **Components:** The component can be of two types-a global variable declaration (**declaration**) or a function(**function**) followed by another series of components.

- **Function:** A function is defined by the series: datatype(**DT**) identifier(**ID**) ‘(‘ **parameter** ‘)’ **block** This follows the general syntax for function signature having the return type followed by function name followed by parameter list in parenthesis and finally the function body as a block of statements.

- **Parameter:** The parameter list for a function can be either empty or more than one parameters (**multiparameter**)

- **Multiparameter:** A multiparameter can be either a single parameter defined by the sequence datatype(**DT**) identifier (**ID**) or a single parameter as defined preceded by **multiparameter** separated by ‘,’.

- **Block:** A block is defined by the following sequence: ‘{‘ **multistatement** ‘}’ ,i.e, a series of statements enclosed in curly braces.

- **Multistatement:** A multistatement is defined either as empty or as a **statement** followed by **multistatement**. It thus gives rise to the sense of a sequence of succeeding statements.

- **Statement:** A statement can be of the following 9 types:

  i. **declaration**:for variable declaration
  ii. **conditional**:for if-else constructs
  iii. **loop**:for while loop construct
  iv. **RETURN ‘;’**:for return from functions without return value
  v. **RETURN expression ‘;’**:for return from functions with return value;

vi. **output**:for output statements
vii. **init_decl**:for assignment statements
viii. **switch**:for switch statements
ix. **functioncall ';'**:for direct functioncalls without storing return values.

- **Declaration:** A declaration can be of two types-Initialized(**DT init_decl**) and uninitialized declaration(**un_init_decl**). The initialized declarartion is basically an assignment statement preceeded by a datatype and hence that structure.

- **Un_init_decl:** An uninitialized declarartion is defined by the sequence – datatype(**DT**) identifier (**ID**) ';'

- **init_decl:** An initialized declarartion is defined by the sequences:

  i. datatype(**DT**) identifier(**ID**) '=' **getvalue** ';'
  ii. datatype(**DT**) identifier(**ID**) **MULA getvalue** ';'
  iii. datatype(**DT**) identifier(**ID**) **DIVA getvalue** ';'
  iv. datatype(**DT**) identifier(**ID**) **SUBA getvalue** ';'
  v. datatype(**DT**) identifier(**ID**) **ADDA getvalue** ';'

- **getvalue:** The value can either be an **INPUT** or a **value.**

- **Value:** A value can be obtained or be of the following types:

i. **boolean**
ii. **functioncall**
iii. **expression**

- **boolean:** The boolean value can either be true(**TR**) or false(**FL**).

- **Functioncall:** A functioncall is defined by the sequence-identifier(**ID**) '(' argument-list(**argument**) ')'

- **argument:** The argument for a function can be either null or a sequence of values(**multiargument**)

- **multiargument:** A multiargument can either be a single value or a sequence of values separated by commas.

- **Expression:** An expression can be of the following types(Note that using the features the associativity and precedence of operators have already been defined as followed by ANSI-C):

i. **expression** '+' **expression**
ii. **expression** '-' **expression**
iii. **expression** '*' **expression**
iv. **expression** '/' **expression**
v. **expression** '|' **expression**
vi. **expression** '&' **expression**
vii. **expression** '^' **expression**
viii. **expression** '@' **expression**
ix. **expression** '~' **expression**

x. **ID**
xi. **NUMBER**

- **conditional:** Two versions of the conditional statements are presently supported:
  i. if-else: This is defined by the sequence: **IF** ‘(‘ **boolexpression** ‘)’ **block ELSE block**
  ii. only if: This is defined by the sequence **IF** ‘(‘ **boolexpression** ‘)’ **block**

- **loop:** Only the while loop construct is defined by the sub-C language under consideration and it is defined by the following sequence: **WHILE** ‘(‘ **boolexpression** ‘)’ **block**

- **boolexpression:**A boolean expression can be of three types:
  i. A sequence of boolean statements(**boolstmt**) separated by logical operators(**LOGOP**)
  ii. A **boolexpression** preceeded by negation.The sequence is: **NOT** ‘(‘ **boolexpression** ‘)’
  iii. Only a simple boolstmt

- **boolstmt:** A boolean statement is defined as either a **boolean** value or two **values** separated by a relational operator(**RELOP**)

- **output:** An output statement is used to write something to the standard output and is defined by the sequenece- **OUTPUT** ‘(‘ **printable** ‘)’ ‘;’

- **printable:** A printable can either be **multiarguments** specifying a series of values to be printed or a **STRING** showing only a simple string to be printed. The sub-C language does not support string type variables and hence character sequences cannot be printed with values of other types.

- **Switch:** A switch statement is defined by the sequence- **SWITCH** ‘(‘ **value** ‘)’ ‘{‘ **cases** ‘}’

- **cases:** Cases can be of three types:
  i. null case
  ii. case with break: defined by the sequence **CASE NUMBER** ‘:’ **multistatement BREAK ‘;’ cases**
  iii. case without break: defined by the sequence **CASE NUMBER** ‘:’ **multistatement cases.**

Note that our sub-C language does not take in expressions or variables as case tags since constant variables are not supported and case tags require constant variables.

Description of classes used to support Grammar Design

This section gives the details of the c++ classes defined in classes.h file which supports the grammar design for all the constructs.

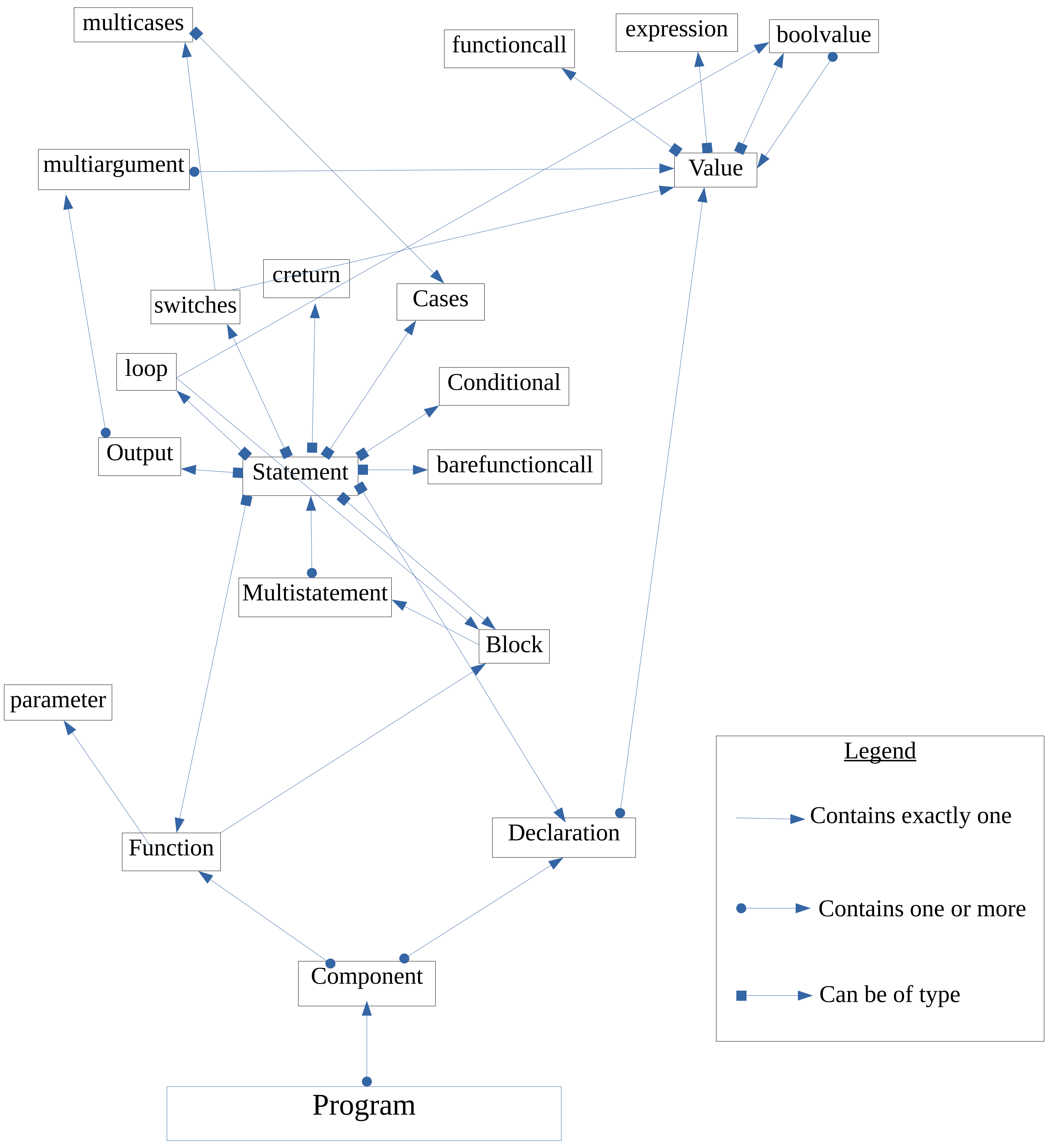
In this project, no three address code is printed while identifying the constructs. Rather a sophisticated structure is created out of the entire parse tree traversal which when printed gives the three address code.

How these structures link together as pieces of a puzzle is what this section explains.

At first a pictoral representation of different classes has been given which shows all the inheritance and containment relations between them.

This diagram is then followed by a walk through of the members and the printing function of each class which basically explains the working of the magic of the three address intermediate code generation from a program file.

Class Relationship Diagram:



Walkthrough of details of each class:

- **Value:**Base class. Contains a variable to store the identifier of the variable to which it evaluates to. Also defines a virtual function getcode to be implemented by the other derived classes.

- **Expression: Derived from Value.** Contains additional flag to mark if it is a terminal symbol or not. Also contains a character string representing the final assignment to be evaluated and two members of expression type representing the two operands.Get code calls getcode functions of the two operands and then prints the evaluatedto members of both of them separated by the code.

- **Functioncall:Derived from value.** Conatins a character string for function name and an array of value type representing the arguments passed while function call. The getcode method first calls getcode() of all the arguments as they are to be evaluated first.Then prints the evaluatedto parameter of the arguments preceeded by "Param" to specify that they are to pushed on to the stack while target code generation. Then the functionname is printed preceeded by "call" specifying to jump to the beginning of the function definition.

- **Multiargument:**A structure containing a vector of value type.

- **Statement:** The base class of all types of statements. Defines the start and end label of every statement and defines a virtual function evaluatestatement() for printing the three address code for the statement.

- **Creturn:Derived from statement.** Defines a return statement. Contains a character string representing the variable name to be returned which can be null for returning nothing. The evaluatestatement prints the start label of the statement followed by the "Return" and return value, ending with the end label of the statement.

- **Output:Derived from statement.** Defines the output statement to be printed to the console. Declares the additional member called m of multiargument type which captures the arguments passed to be printed.The evaluatestatement() function calls the getcode() function for every value type variable in the array inside the vector for m. This is followed by printing "Write" and the evaluatedto member of the same.

- **Declaration: Derived from statement.** Defines a character string id representing the lhs of a declaration. The rhs is of value type.The evaluatestatement() fucntion first prints the start label. This is followed by invokation of the getcode() function of the rhs followed by id '=' and evaluatedto member of rhs.

- **Multistatement:**structure containing an array of statements.

- **Block: Derived from statement.** Contains additional parameter m of multistatement type. Evaluatedto() function first prints the start Then calls evaluatestatement() for every index of the vector of statements in m.

- **Boolvalue:Derived from value.** Contains a character for the operator involved and two value type members representing the two operands. One of them can be null in case of NOT statement.The evaluatedto() function first calls getcode for both the operands followed by assignment of the evaluatedto parameter of itself with the two members separated by the operator.

- **Conditional: Derived from statement.** Defines parameters as flag for elsethere and two blocks representing ifblock and elseblock.evaluatedstatement() function prints the required info by using appropriate methods as defined by the code.

- **Loop:Derived from statement.** Defines a boolvalue representing condition. And the block representing body of the loop. The evaluatedstatement() method prints the same.

- **Parameter:**contains a structure of character strings representing the parameter names in the formal signature of a function definition.

- **Function:Derived from statement.** Defines a character array for the function name, a parameter type member m for the list of parameters and a block type member representing the function body.The evaluatestatement() method prints the same details appropriately.

- **Component:Derived from statement.** Contains an array of statements constituting it.

- **Cases:derived from statement.** Contains a value type member representing the label. And multistatement type member representing the statements to follow and a flag to represent if this is followed by a break statement.

- **Multicases:** conatins an array of cases for a switch statement.

- **Switches:Derived from statement.** Conatins value type member for the switch variable and multicases type member for representing the sequence of cases.

- **Barefunctioncall:Derived from statement.** Represents a functioncall statement where return value need not be stored. Contains just a value type argument for functioncall evaluation.

For further references of implementation details, refer classes.h portion in the code section.

Description of other helper functions

- **getnextlabel():** To generate the next unique label.
- **Getnexttemp()**: To generate the next unique temporary variable name.
- **Getchararray()**: To convert a string to a character array.
- **Createexpression()**: To create the expression type object for evaluating expressions.
- **Getexpressioncode()**:To get the final code of every expression type object.

For further references of implementation details, refer header.h portion in the code section.

<u>CODE</u>

CLASSES.H

```
# include<vector>

using namespace std;


class value
{
      public:
            char* evaluatedto;
            virtual void getcode(){}
};

class expression: public value
{
      public:
            int isterminal;
            char* code;
            expression* cexpression1;
            expression* cexpression2;

            void getcode()
            {

                  if(cexpression1==NULL)
                  return;

                  cexpression1->getcode();
                  cexpression2->getcode();

                  cout<<"\t"<<evaluatedto<<" "<<code<<endl;
            }
};

class functioncall:public value
{
      public:
            char* functionname;
            vector<value*> arguments;

            void getcode()
            {

                  for(int i=0;i<arguments.size();i++)
                  {
                        if(arguments[i]!=NULL)
                        arguments[i]->getcode();

                  }

                  for(int i=0;i<arguments.size();i++)
                  {
                        if(arguments[i]!=NULL)
                        cout<<"\tParam "<<arguments[i]->evaluatedto<<endl;
                  }

                  cout<<"\tcall "<<functionname<<endl;
            }
};
```

```
struct multiargument
{
      vector<value*> subargument;
};

class statement
{
      public:
            char* start;
            char* end;
            virtual void evaluatestatement(){}
};

class creturn:public statement
{
      public:
            char* returnvalue;
            void evaluatestatement()
            {
                  cout<<endl<<start<<":"<<endl;
                  cout<<"\tReturn ";
                  if(returnvalue!=NULL)
                  cout<<returnvalue;

                  cout<<endl;
            }
};

class output:public statement
{
      public:
            multiargument* m;
            void evaluatestatement()
            {
                  if(m->subargument.size()>0)
                  {
                        cout<<endl<<start<<":"<<endl;
                        for(int i=0;i<m->subargument.size();i++)
                        {
                              m->subargument[i]->getcode();
                              cout<<"\tWrite "<<m->subargument[i]-
>evaluatedto<<endl;
                        }
                  }
            }
};


class declaration:public statement
{
      public:
            char* id;
            value* rhs;
            void evaluatestatement()
            {
                  if(rhs!=NULL)
                  {
                        //cout<<"Entered\n";
                        cout<<endl<<start<<":"<<endl;
                        rhs->getcode();
                        cout<<"\t"<<id<<"="<<rhs->evaluatedto<<endl;
                        }
                }
        };

        struct multistatement
        {
                vector<statement*> constituents;
        };

        class block:public statement
        {
                public:
                        multistatement *m;
                        void evaluatestatement()
                        {
                                cout<<endl<<start<<":"<<endl;
                                if(m->constituents.size())
                                {


                                        if(m==NULL)
                                        {

                                                return;
                                        }

                                        for(int i=0;i<m->constituents.size();i++)
                                        {
                                                if(m->constituents[i]!=NULL)
                                                {
                                                        m->constituents[i]->evaluatestatement();
                                                }
                                        }
                                }
                        }
        };

        class boolvalue:public value
        {
                public:
                        char* op;
                        value* v1;
                        value* v2;

                        void getcode()
                        {

                                if(v1!=NULL)
                                v1->getcode();
                                else
                                {
                                        v1=new value();
                                        v1->evaluatedto=new char[1];
                                        v1->evaluatedto[0]='\0';
                                }

                                v2->getcode();

                                cout<<"\t"<<this->evaluatedto<<"="<<v1->evaluatedto<<op<<v2-
        >evaluatedto<<endl;
                        }
        };
```

```
class conditionalt:public statement
{
	public:
		int elsethere;
		boolvalue* condition;
		block* ifblock;
		block* elseblock;

		void evaluatestatement()
		{
			cout<<endl<<start<<":"<<endl;
			condition->getcode();

			if(elsethere)
			{
				cout<<"\tIf "<<condition->evaluatedto<<" is false go to
"" "<<elseblock->start<<endl;
				ifblock->evaluatestatement();
				cout<<"\tGoto "" "<<end<<endl;

				elseblock->evaluatestatement();
				cout<<"\tGoto "" "<<end<<endl;

			}

			else
			{
				cout<<"\tIf "<<condition->evaluatedto<<" is false go to
"" "<<end<<endl;
				ifblock->evaluatestatement();
			}

			cout<<end<<":"<<endl;
		}
};

class loop:public statement
{
	public:
		boolvalue* condition;
		block* body;

		void evaluatestatement()
		{
			cout<<endl<<start<<":"<<endl;
			condition->getcode();

			cout<<"\tIf "<<condition->evaluatedto<<" is false go to "" "
<<end<<endl;

			body->evaluatestatement();

			cout<<"\tReturn to "" "<<start<<endl;

			cout<<end<<":"<<endl;
		}

};

struct parameter
```

```
{
	vector<char*> parameters;
};

class function:public statement
{
	public:

		char* name;
		parameter* p;
		block* b;
		function()
		{
			b=NULL;
		}
		void evaluatestatement()
		{
			cout<<endl<<"\tDefine function "<<name<<endl;
			for(int i=0;i<p->parameters.size();i++)
			if(p->parameters[i]!=NULL)
			cout<<"\tParam "<<p->parameters[i]<<endl;

			if(b!=NULL)
			{

				b->evaluatestatement();
				cout<<end<<":\n Function Completed\n";
			}
			else
			cout<<"\tblock is null\n";
		}
};

class component
{
	public:
		vector<statement*> parts;

		void generatecode()
		{
			for(int i=0;i<parts.size();i++)
			{
				if(parts[i]!=NULL)
				parts[i]->evaluatestatement();
			}
		}
};

class cases:public statement
{
	public:
		value* v;
		multistatement* m;
		int hasbreak;

		void evaluatestatement()
		{
			//cout<<endl<<start<<endl;

			for(int i=0;i<m->constituents.size();i++)
			m->constituents[i]->evaluatestatement();
```

```
                 cout<<end<<":"<<endl;
            }
};

struct multicases
{
      char* start;
      char* end;
      vector<cases*> m;
};

class switches:public statement
{
      public:
            value* v;
            multicases* c;

            void evaluatestatement()
            {
                  cout<<endl<<start<<":"<<endl;
                  v->getcode();
                  int i;
                  for(i=0;i<c->m.size()-1;i++)
                  {
                        cout<<endl<<c->m[i]->start<<":"<<endl;
                        cout<<"\tIf "<<v->evaluatedto<<"!="<<c->m[i]->v-
>evaluatedto<<"goto "<<c->m[i+1]->start<<endl;
                        c->m[i]->evaluatestatement();
                        if(c->m[i]->hasbreak)
                        cout<<"\tGoto "<<end<<endl;
                  }

                  cout<<endl<<c->m[i]->start<<":"<<endl;

                  cout<<"\tIf "<<v->evaluatedto<<"!="<<c->m[i]->v-
>evaluatedto<<"goto "<<end<<endl;
                  c->m[i]->evaluatestatement();



            }
};


class barefunctioncall:public statement
{
      public:
            value* v;
            void evaluatestatement()
            {
                  //cout<<endl<<start<<endl;
                  v->getcode();
            }
};
```

```
}
```

HEADER.H

```
# include<vector>
# include<map>
# include<stack>
int temp_count=1;
int label_count=1;
using namespace std;

string getnextlabel()
{
      string s="";
      int i=label_count;
      label_count++;

      while(i>0)
      {
            s=(char)(i%10+48)+s;
            i/=10;
      }

      s="label"+s;

      cout<<s<<endl;
      return s;
}


string getnexttemp()
{
      string s="";
      int i=temp_count;
      temp_count++;

      while(i>0)
      {
            s=(char)(i%10+48)+s;
            i/=10;
      }
      return "t"+s;
}

char* getchararray(string s)
{
      char *a=new char[s.length()+1];
      for(int i=0;i<s.length();i++)
      a[i]=s[i];

      a[s.length()]='\0';

      return a;
}

expression* createexpression(char c,expression* e1,expression* e2)
{
      expression* e=new expression();
      e->cexpression1=e1;
      e->cexpression2=e2;

      e->evaluatedto=getchararray(getnexttemp());
```

```
	string s1(e1->evaluatedto);
	string s2(e2->evaluatedto);

	string s="="+s1;
	s.push_back(c);
	s+=s2;

	e->code=getchararray(s);

	return e;
}

string getexpressioncode(expression* e)
{
	if(e->cexpression1==NULL)
	return "";

	cout<<"Entered getexpressioncode\n";
	string s1(e->evaluatedto);
	string s2(e->code);

	cout<<s1<<" "<<s2<<endl;

	string s=getexpressioncode(e->cexpression1)+"\n"+getexpressioncode(e-
>cexpression2)+"\n"+s1+s2;

	cout<<s<<endl;

	return s;

}
```

LEX SPECIFICATION

```
%{

# include<iostream>
# include "classes.h"
# include "main.tab.h"

using namespace std;

%}

%%

"case"                 {
                             return CASE;
                       }

"switch"         {
                             return SWITCH;
                       }

":"                    {
                             return ':';
                       }

"int"            {
                             yylval.s=new char[strlen(yytext)+1];
                             for(int i=0;i<strlen(yytext);i++)
                             yylval.s[i]=yytext[i];

                             yylval.s[strlen(yytext)]='\0'; ;cout<<"Came int\n";
                             return DT;
                       }

[",""";""(""")""{""}"]  {

                                         yylval.s=new char[strlen(yytext)+1];
                                         for(int i=0;i<strlen(yytext);i++)
                                         yylval.s[i]=yytext[i];

                                         yylval.s[strlen(yytext)]='\0'; ;
cout<<"Came separator "<<yytext<<endl;
                                         return yytext[0];

                                   }

"="                                {
                                         yylval.s=new char[strlen(yytext)+1];
                                         for(int i=0;i<strlen(yytext);i++)
                                         yylval.s[i]=yytext[i];

                                         yylval.s[strlen(yytext)]='\0'; ;
cout<<"Came assignment "<<endl;
                                         return yytext[0];

                                   }

"*="                         {
                                         yylval.s=new char[strlen(yytext)+1];
```

```
                                                 for(int i=0;i<strlen(yytext);i++)
                                                 yylval.s[i]=yytext[i];

                                                 yylval.s[strlen(yytext)]='\0'; ;
cout<<"Came MULA\n";
                                                 return MULA;

                                           }

"/="                             {
                                                  yylval.s=new char[strlen(yytext)+1];
                                                 for(int i=0;i<strlen(yytext);i++)
                                                 yylval.s[i]=yytext[i];

                                                 yylval.s[strlen(yytext)]='\0'; ;
cout<<"Came DIVA\n";
                                                 return DIVA;

                                           }

"+="                             {

                                                 yylval.s=new char[strlen(yytext)+1];
                                                 for(int i=0;i<strlen(yytext);i++)
                                                 yylval.s[i]=yytext[i];

                                                 yylval.s[strlen(yytext)]='\0'; ;
cout<<"Came ADDA\n";
                                                 return ADDA;

                                           }

"-="                             {

                                                 yylval.s=new char[strlen(yytext)+1];
                                                 for(int i=0;i<strlen(yytext);i++)
                                                 yylval.s[i]=yytext[i];

                                                 yylval.s[strlen(yytext)]='\0'; ;
cout<<"Came SUBA\n";

                                                 return SUBA;

                                           }

"input()"                        {
                                                 yylval.s=new char[strlen(yytext)+1];
                                                 for(int i=0;i<strlen(yytext);i++)
                                                 yylval.s[i]=yytext[i];

                                                 yylval.s[strlen(yytext)]='\0'; ;
cout<<"Came input statement\n";
                                                 return INPUT;

                                           }

"output"                         {
                                                 yylval.s=new char[strlen(yytext)+1];
                                                 for(int i=0;i<strlen(yytext);i++)
                                                 yylval.s[i]=yytext[i];
```

```
                                                yylval.s[strlen(yytext)]='\0'; ;
cout<<"Came output statement\n";
                                                return OUTPUT;

                                          }

"bool"                                    {

                                                yylval.s=new char[strlen(yytext)+1];
                                                for(int i=0;i<strlen(yytext);i++)
                                                yylval.s[i]=yytext[i];

                                                yylval.s[strlen(yytext)]='\0'; ;
cout<<"Came bool\n";
                                                return DT;

                                          }

"true"                                    {
                                                yylval.s=new char[strlen(yytext)+1];
                                                for(int i=0;i<strlen(yytext);i++)
                                                yylval.s[i]=yytext[i];

                                                yylval.s[strlen(yytext)]='\0'; ;
cout<<"Came true\n";
                                                return TR;

                                          }

"false"                                   {
                                                yylval.s=new char[strlen(yytext)+1];
                                                for(int i=0;i<strlen(yytext);i++)
                                                yylval.s[i]=yytext[i];

                                                yylval.s[strlen(yytext)]='\0'; ;
cout<<"Came false\n";
                                                return FL;
                                          }

"if"                                {
                                                yylval.s=new char[strlen(yytext)+1];
                                                for(int i=0;i<strlen(yytext);i++)
                                                yylval.s[i]=yytext[i];

                                                yylval.s[strlen(yytext)]='\0'; ;
cout<<"Came if\n";
                                                return IF;

                                          }

"else"                                    {
                                                yylval.s=new char[strlen(yytext)+1];
                                                for(int i=0;i<strlen(yytext);i++)
                                                yylval.s[i]=yytext[i];

                                                yylval.s[strlen(yytext)]='\0'; ;
cout<<"Came else\n";
                                                return ELSE;

                                          }

"while"                                   {
```

```
                                        yylval.s=new char[strlen(yytext)+1];
                                        for(int i=0;i<strlen(yytext);i++)
                                        yylval.s[i]=yytext[i];

                                        yylval.s[strlen(yytext)]='\0'; ;
cout<<"Came while\n";
                                        return WHILE;
                                  }

"break"                           {
                                        yylval.s=new char[strlen(yytext)+1];
                                        for(int i=0;i<strlen(yytext);i++)
                                        yylval.s[i]=yytext[i];

                                        yylval.s[strlen(yytext)]='\0'; ;
cout<<"Came break\n";
                                        return BREAK;

                                  }

"return"                    {
                                        yylval.s=new char[strlen(yytext)+1];
                                        for(int i=0;i<strlen(yytext);i++)
                                        yylval.s[i]=yytext[i];

                                        yylval.s[strlen(yytext)]='\0'; ;
cout<<"Came return\n";
                                        return RETURN;
                                  }

[0-9]+                            {

                                        yylval.e=new expression();

                                        yylval.e->isterminal=1;
                                        yylval.e->cexpression1=NULL;
                                        yylval.e->cexpression2=NULL;

                                        yylval.e->evaluatedto=new
char[strlen(yytext)+1];
                                        for(int i=0;i<strlen(yytext);i++)
                                        yylval.e->evaluatedto[i]=yytext[i];

                                        yylval.e-
>evaluatedto[strlen(yytext)]='\0';

                                        cout<<"Came number\n";return NUMBER;

                                  }

"<"                               {
                                        yylval.e=new expression();

                                        yylval.e->isterminal=1;
                                        yylval.e->cexpression1=NULL;
                                        yylval.e->cexpression2=NULL;

                                        yylval.e->evaluatedto=new
char[strlen(yytext)+1];
                                        for(int i=0;i<strlen(yytext);i++)
                                        yylval.e->evaluatedto[i]=yytext[i];
                                        yylval.e-
>evaluatedto[strlen(yytext)]='\0';
                                        return RELOP;
                                  }

"<="                        {

                                        yylval.e=new expression();

                                        yylval.e->isterminal=1;
                                        yylval.e->cexpression1=NULL;
                                        yylval.e->cexpression2=NULL;

                                        yylval.e->evaluatedto=new
char[strlen(yytext)+1];
                                        for(int i=0;i<strlen(yytext);i++)
                                        yylval.e->evaluatedto[i]=yytext[i];

                                        yylval.e-
>evaluatedto[strlen(yytext)]='\0';
                                        return RELOP;

                                  }

">"                               {
                                        yylval.e=new expression();

                                        yylval.e->isterminal=1;
                                        yylval.e->cexpression1=NULL;
                                        yylval.e->cexpression2=NULL;

                                        yylval.e->evaluatedto=new
char[strlen(yytext)+1];
                                        for(int i=0;i<strlen(yytext);i++)
                                        yylval.e->evaluatedto[i]=yytext[i];

                                        yylval.e-
>evaluatedto[strlen(yytext)]='\0';
                                        return RELOP;

                                  }

">="                        {

                                        yylval.e=new expression();

                                        yylval.e->isterminal=1;
                                        yylval.e->cexpression1=NULL;
                                        yylval.e->cexpression2=NULL;

                                        yylval.e->evaluatedto=new
char[strlen(yytext)+1];
                                        for(int i=0;i<strlen(yytext);i++)
                                        yylval.e->evaluatedto[i]=yytext[i];

                                        yylval.e-
>evaluatedto[strlen(yytext)]='\0';
                                        return RELOP;

                                  }

"=="                        {
```

```
                                                yylval.e=new expression();

                                                yylval.e->isterminal=1;
                                                yylval.e->cexpression1=NULL;
                                                yylval.e->cexpression2=NULL;

                                                yylval.e->evaluatedto=new
char[strlen(yytext)+1];
                                                for(int i=0;i<strlen(yytext);i++)
                                                yylval.e->evaluatedto[i]=yytext[i];

                                                yylval.e-
>evaluatedto[strlen(yytext)]='\0';
                                                return RELOP;
                                          }

"!="                            {
                                                yylval.e=new expression();

                                                yylval.e->isterminal=1;
                                                yylval.e->cexpression1=NULL;
                                                yylval.e->cexpression2=NULL;

                                                yylval.e->evaluatedto=new
char[strlen(yytext)+1];
                                                for(int i=0;i<strlen(yytext);i++)
                                                yylval.e->evaluatedto[i]=yytext[i];

                                                yylval.e-
>evaluatedto[strlen(yytext)]='\0';
                                                return RELOP;
                                          }

"&&"                            {
                                                yylval.e=new expression();

                                                yylval.e->isterminal=1;
                                                yylval.e->cexpression1=NULL;
                                                yylval.e->cexpression2=NULL;

                                                yylval.e->evaluatedto=new
char[strlen(yytext)+1];
                                                for(int i=0;i<strlen(yytext);i++)
                                                yylval.e->evaluatedto[i]=yytext[i];

                                                yylval.e-
>evaluatedto[strlen(yytext)]='\0';


                                                return LOGOP;

                                          }

"||"                            {
                                                yylval.e=new expression();

                                                yylval.e->isterminal=1;
                                                yylval.e->cexpression1=NULL;
                                                yylval.e->cexpression2=NULL;

                                                yylval.e->evaluatedto=new
char[strlen(yytext)+1];
```

```
                                        for(int i=0;i<strlen(yytext);i++)
                                        yylval.e->evaluatedto[i]=yytext[i];

                                        yylval.e-
>evaluatedto[strlen(yytext)]='\0';

                                        cout<<"Came ||\n";
                                        return LOGOP;
                                  }

"+"                               {
                                        yylval.s=new char[strlen(yytext)+1];
                                        for(int i=0;i<strlen(yytext);i++)
                                        yylval.s[i]=yytext[i];

                                        yylval.s[strlen(yytext)]='\0'; ;
cout<<"Came +\n";
                                        return '+';

                                  }

"-"                               {
                                        yylval.s=new char[strlen(yytext)+1];
                                        for(int i=0;i<strlen(yytext);i++)
                                        yylval.s[i]=yytext[i];

                                        yylval.s[strlen(yytext)]='\0'; ;
cout<<"Came -\n";
                                        return '-';
                                  }

"/"                          {
                                        yylval.s=new char[strlen(yytext)+1];
                                        for(int i=0;i<strlen(yytext);i++)
                                        yylval.s[i]=yytext[i];

                                        yylval.s[strlen(yytext)]='\0'; ;
cout<<"Came /\n";
                                        return '/';
                                  }

"*"                               {
                                        yylval.s=new char[strlen(yytext)+1];
                                        for(int i=0;i<strlen(yytext);i++)
                                        yylval.s[i]=yytext[i];

                                        yylval.s[strlen(yytext)]='\0'; ;
cout<<"Came *\n";
                                        return '*';

                                  }

"&"                               {
                                        yylval.s=new char[strlen(yytext)+1];
                                        for(int i=0;i<strlen(yytext);i++)
                                        yylval.s[i]=yytext[i];

                                        yylval.s[strlen(yytext)]='\0'; ;
cout<<"Came &\n";
                                        return '&';
                                  }
```

```
"|"                            {
                                          yylval.s=new char[strlen(yytext)+1];
                                          for(int i=0;i<strlen(yytext);i++)
                                          yylval.s[i]=yytext[i];

                                          yylval.s[strlen(yytext)]='\0'; ;
cout<<"Came |\n";
                                          return '|';
                                     }

"^"                                  {

                                          yylval.s=new char[strlen(yytext)+1];
                                          for(int i=0;i<strlen(yytext);i++)
                                          yylval.s[i]=yytext[i];

                                          yylval.s[strlen(yytext)]='\0'; ;
cout<<"Came ^\n";
                                          return '^';

                                     }

"@"                                  {
                                          yylval.s=new char[strlen(yytext)+1];
                                          for(int i=0;i<strlen(yytext);i++)
                                          yylval.s[i]=yytext[i];

                                          yylval.s[strlen(yytext)]='\0'; ;
cout<<"Came @\n";
                                          return '@';
                                     }

"~"                                  {

                                          yylval.s=new char[strlen(yytext)+1];
                                          for(int i=0;i<strlen(yytext);i++)
                                          yylval.s[i]=yytext[i];

                                          yylval.s[strlen(yytext)]='\0'; ;
cout<<"Came ~\n";
                                          return '~';
                                     }

"\!"                           {
                                          yylval.s=new char[strlen(yytext)+1];
                                          for(int i=0;i<strlen(yytext);i++)
                                          yylval.s[i]=yytext[i];

                                          yylval.s[strlen(yytext)]='\0'; ;
cout<<"Came !\n";
                                          return NOT;

                                     }

"%"                                  {

                                          yylval.s=new char[strlen(yytext)+1];
                                          for(int i=0;i<strlen(yytext);i++)
                                          yylval.s[i]=yytext[i];

                                          yylval.s[strlen(yytext)]='\0'; ;
cout<<"Came %\n";
```

```
                                   return '%';
                              }

"\""[A-Za-z0-9\\ =\*^&%$#@!,.;:]*"\""  {

                                                              yylval.e=new
expression();

                                                              yylval.e-
>isterminal=1;
                                                              yylval.e-
>cexpression1=NULL;
                                                              yylval.e-
>cexpression2=NULL;

                                                              yylval.e-
>evaluatedto=new char[strlen(yytext)+1];
                                                              for(int
i=0;i<strlen(yytext);i++)
                                                              yylval.e-
>evaluatedto[i]=yytext[i];
      yylval.e->evaluatedto[strlen(yytext)]='\0';
                                             cout<<"Came STRING"<<yytext<<endl;

                                                              return STRING;
                                                   }

[A-Za-z_][A-Za-z_0-9]*                        {
                                                              yylval.e=new
expression();

                                                              yylval.e-
>isterminal=1;
                                                              yylval.e-
>cexpression1=NULL;
                                                              yylval.e-
>cexpression2=NULL;
                                                              yylval.e-
>evaluatedto=new char[strlen(yytext)+1];
                                                              for(int
i=0;i<strlen(yytext);i++)
                                                              yylval.e-
>evaluatedto[i]=yytext[i];

                                                              yylval.e-
>evaluatedto[strlen(yytext)]='\0';

                                                              cout<<"Came ID
"<<yytext<<endl;

                                                              return ID;

                                                   }

"%%"        cout<<"Came EOF\n";return EF;

[ \t\n]+    {}

.           { cout<<yytext<<" &&\n";;}

%%
```

YACC SPECIFICATION

```
%{
# include<iostream>
# include<fstream>
# include<cstring>
# include "classes.h"
# include "header.h"
using namespace std;

int yylex();
void yyerror(const char*);

string finalcode="";
string expressioncode="";
string statementcode="";


%}


%union
{
      char* s;
      int a;
      value* v;
      expression* e;
      functioncall* fc;
      multiargument* m;
      statement* st;
      output* o;
      declaration* d;
      multistatement* ms;
      block* b;
      boolvalue* bv;
      conditionalt* c;
      loop * l;
      parameter* p;
      function* f;
      component* cp;
      switches* sw;
      multicases* cs;
      barefunctioncall* bf;
}


%token  EF DT TR FL IF ELSE WHILE RELOP LOGOP ID NUMBER NOT INPUT OUTPUT MOD
STRING BREAK RETURN
%token  MULA ADDA DIVA SUBA SWITCH CASE

%left  '|' '^' '&' '~'
%left  '+' '-'
%left  '*' '/'
%left '%'
%right  '@'

%type <s>  EF DT TR FL IF ELSE WHILE  NOT INPUT OUTPUT MOD  BREAK RETURN
```

```
%type <s> MULA ADDA DIVA SUBA

%type <s> '|' '^' '&' '~'
%type <s> '+' '-'
%type <s> '*' '/'
%type <s>'%'
%type <s> '@'
%type <s> '=' ';' "(" ")" ","

%type <cp> program component
%type <f> function
%type <fc> functioncall
%type <p> parameter multiparameter
%type <e> expression ID NUMBER STRING LOGOP RELOP
%type <st> statement
%type <d> declaration init_decl un_init_decl
%type <c> conditional
%type <l> loop
%type <o> output
%type <m> multiargument printable argument
%type <ms> multistatement
%type <b> block
%type <v> value getvalue
%type <bv> boolexpression boolstmt boolean
%type <sw> switch
%type <cs> cases

%%

program:component EF  {
                                        cout<<"Program detected
successfully\n";


     cout<<"**************************************************************\n";
                                        cout<<"SECTION-
II\n-------------------------------------------->\n";
                                        $1->generatecode();

                                  }

//--------------------------------------------------------------------------------
--------------------

component:      {
                            $$=new component();

                       }
|function component    {
                                        cout<<"Function detected1
successfully\n";
                                        $$=new component();

                                        $$->parts.push_back($1);

                                        for(int i=0;i<$2->parts.size();i++)
                                        $$->parts.push_back($2->parts[i]);

                                        cout<<"Function componenet generated
successfully\n";
                                  }
|declaration component      {
```

```
                                        cout<<"Declaration detected
successfully\n";
                                        $$=new component();

                                        $$->parts.push_back($1);

                                        $1->evaluatestatement();

                                        for(int i=0;i<$2->parts.size();i++)
                                        $$->parts.push_back($2->parts[i]);

                                        cout<<"Declaration code generated
successfully\n";

                                  }
;

//-----------------------------------------------------------------------------
--------------------


parameter:                              {
                                                    cout<<"No parameter detected
successfully\n";

                                                    $$=new parameter;
                                              }

|multiparameter                         {
                                                    cout<<"Function parameter
detected succesfully\n";

                                              }
;

//-----------------------------------------------------------------------------
--------------------


multiparameter:multiparameter ',' DT ID {


     cout<<"Multiparameter detected successfully\n";

                                                                $$=new parameter;

                                                                for(int i=0;i<$1-
>parameters.size();i++)
                                                                $$-
>parameters.push_back($1->parameters[i]);

                                                                $$-
>parameters.push_back($4->evaluatedto);

                                                           }
|DT ID          {
                      cout<<"One parameter detected successfully\n";

                      $$=new parameter;

                      $$->parameters.push_back($2->evaluatedto);
```

```
                 }
;

//--------------------------------------------------------------------------------
--------------------


function:DT ID '(' parameter ')' block {

                                       cout<<"function detected
successfully\n";

                                       $$=new function();

                                       $$->name=$2->evaluatedto;
                                       $$->p=$4;
                                       $$->b=$6;

                                       if($6!=NULL)
                                       $$->end=$6->end;

                                       cout<<"Function code generated
successfully\n";
                                       }
;

//--------------------------------------------------------------------------------
--------------------

expression:expression '+' expression{
                                                        cout<<"ADD parameter
detected successfully\n";

                                                        $
$=createexpression('+',$1,$3);
                                                   }
|expression '-' expression  {
                                            cout<<"SUB parameter detected
successfully\n";
                                            $$=createexpression('-',$1,$3);
                                       }

|expression '/' expression  {

                                            cout<<"DIV parameter detected
successfully\n";
                                            $$=createexpression('/',$1,$3);

                                       }

|expression '*' expression  {
                                            cout<<"MUL parameter detected
successfully\n";
                                            $$=createexpression('*',$1,$3);
                                       }

|expression '|' expression  {

                                            cout<<"OR parameter detected
successfully\n";
                                            $$=createexpression('|',$1,$3);
```

```
                                        }
    |expression '&' expression  {
                                             cout<<"AND parameter detected
    successfully\n";
                                             $$=createexpression('&',$1,$3);
                                        }
    |expression '^' expression  {
                                             cout<<"XOR parameter detected
    successfully\n";
                                             $$=createexpression('^',$1,$3);
                                        }
    |expression '@' expression  {
                                             cout<<"EXP detected
    successfully\n";
                                             $$=createexpression('@',$1,$3);
                                        }
    |expression '~' expression  {
                                             cout<<"Not detected
    successfully\n";
                                             $$=createexpression('~',$1,$3);
                                        }
    |expression '%' expression  {
                                             cout<<"MOD detecte    d
    successfully\n";
                                             $$=createexpression('%',$1,$3);
                                        }
    |NUMBER                             {
                                             cout<<"Number detected
    successfully "<<$$<<endl;
                                        }
    |ID                                 {
                                             $$->getcode();
                                             cout<<"ID detected successfully
    "<<$$<<endl;
                                        }
    ;
    //--------------------------------------------------------------------------------------------------------
    statement:declaration       {
                                             cout<<"Declaration detected
    successfully\n";
                                             $$=$1;
```

```
                                          }

|conditional                         {
                                               cout<<"Conditional detected
successfully\n";
                                               $$-
>start=getchararray(getnextlabel());
                                               $$-
>end=getchararray(getnextlabel());
                                          }

|loop                                {
                                               cout<<"Loop detected
succesfully\n";
                                               $$-
>start=getchararray(getnextlabel());
                                               $$-
>end=getchararray(getnextlabel());
                                          }

|RETURN ';'                          {
                                               creturn *r=new creturn();
                                               r->returnvalue=NULL; $$=r;
                                               $$-
>start=getchararray(getnextlabel());
                                               $$-
>end=getchararray(getnextlabel());
                                          }

|RETURN expression ';'                    {

                                                         creturn *r=new
creturn();
                                                         r->returnvalue=$2-
>evaluatedto;
                                                         $$=r;
                                                         $$-
>start=getchararray(getnextlabel());
                                                         $$-
>end=getchararray(getnextlabel());
                                                    }

|output                                   {
                                               cout<<"Output statement
detected\n";
                                               $$-
>start=getchararray(getnextlabel());
                                               $$-
>end=getchararray(getnextlabel());
                                          }

|init_decl                      {
                                          cout<<"Assignment statement
encountered\n";
                                          $$->start=getchararray(getnextlabel());
                                          $$->end=getchararray(getnextlabel());
                                     }


|switch                              {
```

```
                                        cout<<"Switch statement encountered\n";
                                        $$->start=getchararray(getnextlabel());

                                  }

|functioncall ';'           {
                                        barefunctioncall* b=new
barefunctioncall();
                                        b->v=$1;
                                        $$=b;
                                        $$->start=getchararray(getnextlabel());
                                        $$->end=getchararray(getnextlabel());
                                  }

;

//-------------------------------------------------------------------------------
---------------------

switch:SWITCH '(' value ')' '{' cases '}'    {

     cout<<"switch identified\n";

                                                                $$=new
switches();
                                                                $$->v=$3;
                                                                $$->c=$6;

                                                                $$->end=$6-
>m[$6->m.size()-1]->end;


     cout<<"SWITCH statement code generated\n";
                                                          }

cases:           {
                      $$=new multicases;
                      $$->start=getchararray(getnextlabel());
                      $$->end=getchararray(getnextlabel());
                      cout<<"Empty case identified\n";

                      cout<<$$->start<<" "<<$$->end<<endl;
                 }

|CASE NUMBER ':' multistatement BREAK ';' cases    {

     cout<<"Case with break identified\n";

                                                                      $$=new
multicases;
                                                                      cases
*i=new cases();
                                                                      i-
>v=$2;
                                                                      i-
>m=$4;
                                                                      i-
>hasbreak=1;
                                                                      i-
>start=getchararray(getnextlabel());
                                                                      i-
>end=getchararray(getnextlabel());
```

```
                                                                              int
k=0;

                                                                              $$-
>m.push_back(i);


     for(int i=0;i<$7->m.size();i++)
                                                                              $$-
>m.push_back($7->m[i]);


     cout<<"case with break code generated\n";
                                                                       }

|CASE NUMBER ':' multistatement cases  {
                                                             cout<<"Case
without break identified\n";

                                                             $$=new
multicases;

                                                             cases *i=new
cases();
                                                             i->v=$2;
                                                             i->m=$4;
                                                             i-
>start=getchararray(getnextlabel());
                                                             i-
>end=getchararray(getnextlabel());


                                                             $$->start=i-
>start;
                                                             $$->end=i->end;

                                                             cout<<$$-
>start<<" "<<$$->end<<endl;

                                                             $$-
>m.push_back(i);

                                                             for(int i=0;i<$5-
>m.size();i++)
                                                             $$-
>m.push_back($5->m[i]);
                                                       }
;
//--------------------------------------------------------------------------------
---------------------

output:OUTPUT'('printable')'';'  {
                                                   cout<<"Output to be
printed\n";
                                                   $$=new output();
                                                   $$->m=$3;

                                              }

;
```

```
//----------------------------------------------------------------------------
--------------------

printable:multiargument     {
                                       cout<<"Multiple arguments to be
printed\n";
                                  }

|STRING                           {
                                       $$=new multiargument;
                                       $$->subargument.push_back($1);

                                  }
;

//----------------------------------------------------------------------------
--------------------

multistatement:       {
                                  $$=new multistatement;

                                  cout<<"No statement detected successfully\n";
                            }
|statement multistatement   {
                                            cout<<"Multistatement detected
successfully\n";


                                            $$=new multistatement;

                                            if($1!=NULL);
                                            $$->constituents.push_back($1);

                                            if($2!=NULL && $2-
>constituents.size()!=0)
                                            {

                                                 int flag=0;

                                                 for(int i=0;i<$2-
>constituents.size();i++)
                                                 {
                                                      if($1!=NULL && $2-
>constituents[i]!=NULL && !flag)
                                                      {
                                                           $2-
>constituents[i]->start=$1->end;

                                                           flag=1;
                                                      }


                                                      $$-
>constituents.push_back($2->constituents[i]);
                                                 }
                                            }


                                       }
;

//----------------------------------------------------------------------------
--------------------
```

```
block:'{' multistatement '}'{

                                                    $$=new block();
                                                    $$-
>start=getchararray(getnextlabel());
                                                    $$-
>end=getchararray(getnextlabel());

                                                    $$->m=$2;

                                                    cout<<"Block detected
successfully\n";


                                              }
;

//--------------------------------------------------------------------------------
--------------------

declaration:un_init_decl    {
                                              cout<<"Uninitialized detected
successfully\n";
                                        }

|DT init_decl                 {
                                              cout<<"Initialized detected
successfully\n";$$=$2;
                                              $$-
>start=getchararray(getnextlabel());
                                              $$-
>end=getchararray(getnextlabel());
                                        }
;

//--------------------------------------------------------------------------------
--------------------

un_init_decl:DT ID ';'      {     $$=new declaration();
                                              $$-
>start=getchararray(getnextlabel());
                                              $$-
>end=getchararray(getnextlabel());
                                              cout<<"Uninitialized detected
successfully\n";

                                        }
;

//--------------------------------------------------------------------------------
--------------------

init_decl:  ID '=' getvalue ';'   {cout<<"Normal initialized detected
successfully\n";

                                              $$=new declaration();
                                              $$->id=$1->evaluatedto;
                                              $$->rhs=$3;

                                              }

| ID MULA getvalue ';'      {
```

```
                                                cout<<"MUL initialized detected
successfully\n";

                                                $$=new declaration;
                                                $$->id=$1->evaluatedto;

                                                $$->rhs=createexpression('*',
(expression*)$1,(expression*)$3);

                                          }

| ID DIVA getvalue ';'      {
                                                cout<<"DIV initialized detected
successfully\n";

                                                $$=new declaration;
                                                $$->id=$1->evaluatedto;

                                                $$->rhs=createexpression('/',
(expression*)$1,(expression*)$3);

                                          }

| ID ADDA getvalue ';'      {
                                                cout<<"ADD initialized detected
successfully\n";

                                                $$=new declaration;
                                                $$->id=$1->evaluatedto;

                                                $$->rhs=createexpression('+',
(expression*)$1,(expression*)$3);

                                          }

| ID SUBA getvalue ';'      {
                                                cout<<"SUB initialized detected
successfully\n";

                                                $$=new declaration;
                                                $$->id=$1->evaluatedto;

                                                $$->rhs=createexpression('-',
(expression*)$1,(expression*)$3);


                                          }
;

//--------------------------------------------------------------------------------
--------------------

getvalue:INPUT                            {
                                                      cout<<"Input detected
successfully\n";

                                                      $$=new value();
                                                      $$->evaluatedto="input\0";

                                                }

|value                                          {
```

```
                                                cout<<"Other value has been
requested\n";
                                           }
;

//------------------------------------------------------------------------------
---------------------

value:boolean                          {
                                                cout<<"Boolean detected
successfully\n";
                                                $$=new value();
                                                $$->evaluatedto=$1-
>evaluatedto;
                                           }

|functioncall                          {
                                                $$=$1;
                                                $$->evaluatedto="Return
Value\n";
                                                cout<<"Function call
detected successfully\n";
                                           }

|expression                            {
                                                $$=$1;
                                                $1->getcode();
                                                cout<<"Expression detected
successfully\n";

                                           }
;


boolean:TR                             {
                                                $$=new boolvalue();
                                                $$->evaluatedto="true\0";
                                                cout<<"True detected
successfully\n";
                                           }
|FL                                        {
                                                $$=new boolvalue();
                                                $$->evaluatedto="false\0";
                                                cout<<"False detected
successfully\n";
                                           }
;

//------------------------------------------------------------------------------
---------------------

functioncall: ID '(' argument ')' {
                                                     cout<<"Function call 2
detected successfully\n";

                                                     $$=new functioncall();
                                                     $$->functionname=$1-
>evaluatedto;
                                                     $$->arguments=$3-
>subargument;
```

```
                                                        cout<<"Function call
code generated\n";
                                                  }
;

argument:             {
                                 $$=new multiargument;
                                 statementcode+="result=returnvalue\n";
                                 cout<<"No argument detected successfully\n";
                           }

|multiargument        {
                                 cout<<"Multiargument detected

successfully\n";
                           }
;

//------------------------------------------------------------------------------
---------------------

multiargument:value              {
                                            cout<<"ID detected

successfully\n";

                                            $$=new multiargument;

                                            $$->subargument.push_back($1);

                                            cout<<"Only value for
multiargument detected\n";
                                       }

|value ',' multiargument    {

                                            cout<<"Multiargument detected
successfully\n";

                                            $$=new multiargument;
                                            $$->subargument.push_back($1);

                                            for(int i=0;i<$3-
>subargument.size();i++)
                                            $$->subargument.push_back($3-

>subargument[i]);

                                            cout<<"Multiargument for function
found\n";
                                       }
;

//------------------------------------------------------------------------------
---------------------

conditional:IF '(' boolexpression ')' block ELSE block  {
                                                              cout<<"If else
detected successfully\n";

                                                              $$=new

conditionalt();
                                                              $$->elsethere=1;
                                                              $$->condition=$3;
                                                              $$->ifblock=$5;
```

```
                                                               $$->elseblock=$7;

                                                               cout<<"If else
code generated successfully\n";
                                                               }
|IF '(' boolexpression ')' block       {

                                                               $$=new
conditionalt();
                                                               $$->elsethere=0;
                                                               $$->condition=$3;
                                                               $$->ifblock=$5;


                                                         }
;

//--------------------------------------------------------------------------------
---------------------

loop: WHILE'('boolexpression')'block   {
                                                               cout<<"Loop
detected successfully\n";

                                                               $$=new loop();
                                                               $$->condition=$3;
                                                               $$->body=$5;

                                                               cout<<"Loop code
generated \n";

                                                         }
;

//--------------------------------------------------------------------------------
---------------------

boolexpression:boolstmt LOGOP boolexpression {

      cout<<"Bool expression detected successfully\n";

                                                                     $$=new
boolvalue();
                                                                     $$-
>evaluatedto=getchararray(getnexttemp());
                                                                     $$->op=$2-
>evaluatedto;
                                                                     $$->v1=$1;
                                                                     $$->v2=$3;


      cout<<"Bool expression code generated\n";
                                                                     }
|NOT '(' boolexpression ')' {
                                                   cout<<"Not detected
successfully\n";
                                                   $$=new boolvalue();
                                                   $$-
>evaluatedto=getchararray(getnexttemp());
                                                   char a[]="NOT\0";
                                                   $$->op=a;
```

```
                                                            $$->v1=NULL;
                                                            $$->v2=$3;

                                                            cout<<"Not code 
generated\n";
                                                      }

|boolstmt                                       {
                                                            cout<<"Bool stmt detected 
successfully\n";
                                                      }

;

//---------------------------------------------------------------------------------------------
---------------------

boolstmt: value RELOP value       {
                                                            cout<<"Bool stmt detected 
successfully\n";

                                                            $$=new boolvalue();
                                                            $$-
>evaluatedto=getchararray(getnexttemp());
                                                            $$->op=$2->evaluatedto;
                                                            $$->v1=$1;
                                                            $$->v2=$3;
                                                            $1->getcode();
                                                            $$->getcode();

                                                            cout<<"Bool statement code 
generated\n";
                                                      }

|boolean                                        {
                                                            cout<<"Came for only one 
plain boolean statement\n";
                                                            $$=$1;
                                                            cout<<"Created code for 
plain boolean statement\n";
                                                      }
;
//---------------------------------------------------------------------------------------------
---------------------
%%

void yyerror(const char * s)
{
      cout<<s<<endl;
}

int main()
{
      cout<<"Parsing of sub c language will begin.\n";
      cout<<"This has two sections\nThe first section elaborates how each 
construct is identified as a bottom up traversal of the inbulit yacc parse 
tree\nThe second section will thereafter provide the three address intermediate 
code\n";


cout<<"*******************************************************************************
*********\n";
```

```
      cout<<"SECTION-I\n------------------------------------------->\n";
      yyparse();


cout<<"***************************************************************************
***********\n";
      cout<<"This concludes the semester assignment of generating three address
intermediate code for a custom sub c language having selected features\n";

}
```

Test Cases

Test Case 1:

Input:

```
int isarmstrong(int n)
{
     int c=n;
     int a;
     int b=0;

     while(n>0)
     {
          a=n%10;
          b+=a*a*a;
          n/=10;
          int p=a*b+9-c;
     }

     if(c==n)
     {
          output("IS ARMSTRONG\n");
     }

     else
     {
          output("NOT ARMSTRONG\n");
     }

}
%%
```

Output:

```
     Define function isarmstrong
     Param n

label29:

label1:
     c=n

label4:
     b=0

label6:
     t1=n>0
     If t1 is false go to  label18

label15:

label7:
     t2 =n%10
     a=t2
```

```
label8:
     t3 =a*a
     t4 =t3*a
     t5 =b+t4
     b=t5

label10:
     t6 =n/10
     n=t6

label12:
     t7 =a*b
     t8 =t7+9
     t9 =t8-c
     p=t9
     Return to  label6
label18:

label18:
     t10=c==n
     If t10 is false go to  label25

label21:

label19:
     Write "IS ARMSTRONG\n"
     Goto  label28

label25:

label23:
     Write "NOT ARMSTRONG\n"
     Goto  label28
label28:
label30:
 Function Completed
```

Test Case 2

Input:

```
int isarmstrong(int n)
{
     int c=n;
     int a;
     int b=0;

     while(n>0)
     {
          a=n%10;
          b+=a*a*a;
          n/=10;
          int p=a*b+9-c;
     }

     if(c==n)
     {
          output("IS ARMSTRONG\n");
     }

     else
     {
          output("NOT ARMSTRONG\n");
     }

}

int isprime(int n)
{
     int k=1;

     while(k<n)
     {
          if(n%k==0)
          {
               output("NOT A PRIME\n");
               return;
          }
          k=k+1;
     }

     output("IS PRIME\n");

}

int ismagic(int n)
{
     while(n>10)
     {
          int k=n;
          int c=0;
          while(k>0)
          {
               c=c+k%10;
               k/=10;
          }

          n=c;
     }

     if(c==1)
     {
          output("IS MAGIC\n");
     }

     else
     {
          output("IS NOT MAGIC\n");
     }
}

int isfactor(int n,int k)
{
     if(n%k==0)
     {
          output("The number is a factor\n");
     }

     else
     {
          int gf6=789-675;
          output("The number is not a factor\n");
          output("Remainder=");
          output(n%k);
          output(n,gf6);
     }
}

int main()
{
     output("Enter 1 to check armstrong, 2 to check prime, 3 to check
magic and 4 to check factor\n");
     int a=input();
     int n=input();

     if(a==1)
     {
          isarmstrong(n);
     }

     else
     {
          if(a==2)
          {
               isprime(n);
          }

          else
          {
               if(a==3)
               {
                    ismagic(n);
               }

               else
               {
                    int k=input();
                    isfactor(n,k);
               }
          }
     }
}

%%
```

Output:

```
     Define function isarmstrong
     Param n

label29:

label1:
     c=n

label4:
     b=0

label6:
     t1=n>0
     If t1 is false go to  label18

label15:

label7:
     t2 =n%10
     a=t2

label8:
     t3 =a*a
     t4 =t3*a
     t5 =b+t4
     b=t5

label10:
     t6 =n/10
     n=t6
```

```
label12:
     t7 =a*b
     t8 =t7+9
     t9 =t8-c
     p=t9
     Return to  label6
label18:

label18:
     t10=c==n
     If t10 is false go to  label25

label21:

label19:
     Write "IS ARMSTRONG\n"
     Goto  label28

label25:

label23:
     Write "NOT ARMSTRONG\n"
     Goto  label28
label28:
label30:
 Function Completed

     Define function isprime
     Param n

label49:

label31:
     k=1

label32:
     t11=k<n
     If t11 is false go to  label46

label43:

label39:
     t12 =n%k
     t13=t12==0
     If t13 is false go to  label40

label37:

label33:
     Write "NOT A PRIME\n"

label34:
     Return
```

```
label40:

label40:
     t14 =k+1
     k=t14
     Return to  label32
label46:

label46:
     Write "IS PRIME\n"
label50:
 Function Completed

     Define function ismagic
     Param n

label79:

label67:
     t15=n>10
     If t15 is false go to  label68

label65:

label51:
     k=n

label52:
     c=0

label54:
     t16=k>0
     If t16 is false go to  label62

label59:

label55:
     t17 =k%10
     t18 =c+t17
     c=t18

label56:
     t19 =k/10
     k=t19
     Return to  label54
label62:

label62:
     n=c
     Return to  label67
label68:

label68:
     t20=c==1
     If t20 is false go to  label75

label71:

label69:
     Write "IS MAGIC\n"
     Goto  label78

label75:

label73:
     Write "IS NOT MAGIC\n"
     Goto  label78
label78:
label80:
 Function Completed

     Define function isfactor
     Param n
     Param k

label99:

label97:
     t21 =n%k
     t22=t21==0
     If t22 is false go to  label95

label83:

label81:
     Write "The number is a factor\n"
     Goto  label98

label95:

label85:
     t23 =789-675
     gf6=t23

label86:
     Write "The number is not a factor\n"

label88:
     Write "Remainder="

label90:
     t24 =n%k
     Write t24

label92:
     Write n
     Write gf6
     Goto  label98
```

```
label98:
label100:
 Function Completed

     Define function main

label135:

label101:
     Write "Enter 1 to check armstrong, 2 to check prime, 3 to check
magic and 4 to check factor\n"

label102:
     a=input

label104:
     n=input

label106:
     t25=a==1
     If t25 is false go to  label131

label109:
     Param n
     call isarmstrong
     Goto  label134

label131:

label129:
     t26=a==2
     If t26 is false go to  label127

label113:
     Param n
     call isprime
     Goto  label130

label127:

label125:
     t27=a==3
     If t27 is false go to  label123

label117:
     Param n
     call ismagic
     Goto  label126

label123:

label119:
     k=input
     Param n
     Param k
     call isfactor
     Goto  label126
label126:
     Goto  label130
label130:
     Goto  label134
label134:
label136:
 Function Completed
```

Test Case 3

Input:

```
int main()
{
     int a=input();
     int b=a+b-56*76;
     output(b);
     if(b>c)
     {
          output("Hello World\n");
     }

     if(b>10 && b<50 && b>18)
     {
          output("ksjfksdf\n");
     }

     else
     {
          output("greatedjads\n");
     }

     while(b>10)
     {
          output("kajdkadjkas\n");
     }

     int t=funct(1,2,3);
     return 0;
}
%%
```

Output:

```
     Define function main

label33:

label1:
     a=input

label2:
     t1 =a+b
     t2 =56*76
     t3 =t1-t2
     b=t3

label4:
     Write b
```

```
label6:
     t4=b>c
     If t4 is false go to  label12

label9:

label7:
     Write "Hello World\n"
label12:

label12:
     t5=b>10
     t6=b<50
     t7=b>18
     t8=t6&&t7
     t9=t5&&t8
     If t9 is false go to  label19

label15:

label13:
     Write "ksjfksdf\n"
     Goto  label22

label19:

label17:
     Write "greatedjads\n"
     Goto  label22
label22:

label22:
     t10=b>10
     If t10 is false go to  label28

label25:

label23:
     Write "kajdkadjkas\n"
     Return to  label22
label28:

label28:
     Param 1
     Param 2
     Param 3
     call funct
     t=Return Value


label30:
     Return 0
label34:
 Function Completed
```

Test Case 4

Input:

```
int main()
{
     int i=5*7-b;
     switch(i)
     {
          case 1:
               int k=354;
               if(k>89)
               {
                    output("Hello World\n");
               }
               else
               {
                    output("Goodbye World\n");
               }
          break;

          case 2:
               int c=100;

               while(c>0)
               {
                    output(c);
                    c-=1;
               }

          case 3:
               int p=190;

               if(p==10)
               {
                    output("To be continued\n");
               }
     }
}
%%
```

Output:

```
     Define function main

label42:

label1:
     t1 =5*7
     t2 =t1-b
     i=t2
```

```
label2:

label39:
     If i!=1goto label37

label3:
     k=354

label4:
     t3=k>89
     If t3 is false go to  label11

label7:

label5:
     Write "Hello World\n"
     Goto  label14

label11:

label9:
     Write "Goodbye World\n"
     Goto  label14
label14:
label40:
     Goto label36

label37:
     If i!=2goto label35

label15:
     c=100

label16:
     t4=c>0
     If t4 is false go to  label24

label21:

label17:
     Write c

label18:
     t5 =c-1
     c=t5
     Return to  label16
label24:
label38:

label35:
     If i!=3goto label36

label25:
     p=190
```

```
label26:
     t6=p==10
     If t6 is false go to  label32

label29:

label27:
     Write "To be continued\n"
label32:
label36:
label43:
 Function Completed
```

Test Case 5

Input:

```
int a=5;

int main()
{
	if(a==10)
	{
		if(b==10)
		{
			if(c==45)
			{
				int k=input();
			}
			else
			{
				int k=armstrong(10,20,secondarmstrong());
			}
		}
	}


}

int b=10;

%%
```

Output:

```
label1:
	a=5

	Define function main

label21:

label19:
	t1=a==10
	If t1 is false go to  label20

label17:

label15:
	t2=b==10
	If t2 is false go to  label16

label13:

label11:
	t3=c==45
```

```
     If t3 is false go to  label9

label5:

label3:
     k=input
     Goto  label12

label9:

label7:
     call secondarmstrong
     Param 10
     Param 20
     Param Return Value

     call armstrong
     k=Return Value

     Goto  label12
label12:
label16:
label20:
label22:
 Function Completed

label23:
     b=10
```

Test Case 6

Input:
```
int main()
{
     int s=10;
     int t;
     switch(s)
     {
          case 1:
               int k=10;
               while(k>0)
               {
                    output(k);
                    output("Hello World\n");
                    k-=1;
               }
          break;

          case 2:

               int p;
               int c=10;


               if(!(c>d && t<10))
               {
                    p=6^8;
               }

               else
               {
                    bool k=true;
               }
          break;

          case 3:

               int d=976;
     }


}

%%
```

Output:

```
     Define function main

label42:
```

```
label1:
     s=10

label4:

label39:
     If s!=1goto label37

label5:
     k=10

label6:
     t1=k>0
     If t1 is false go to  label16

label13:

label7:
     Write k

label8:
     Write "Hello World\n"

label10:
     t2 =k-1
     k=t2
     Return to  label6
label16:
label40:
     Goto label36

label37:
     If s!=2goto label35

label18:
     c=10

label20:
     t3=c>d
     t4=t<10
     t5=t3&&t4
     t6=NOTt5
     If t6 is false go to  label27

label23:

label21:
     t7 =6^8
     p=t7
     Goto  label30

label27:

label25:
     k=true
     Goto  label30
label30:
label38:
     Goto label36

label35:
     If s!=3goto label36

label31:
     d=976
label36:
label43:
 Function Completed
```

Test Case 7

Input:

```
int main()
{
     int c;
     if(t>10)
     {
     }
     else
     {
     }

     int d;

     int p=10;

     output("Abcd");
}

%%
```

Output:

```


     Define function main

label15:

label2:
     t1=t>10
     If t1 is false go to  label5

label3:
     Goto  label8

label5:
     Goto  label8
label8:

label10:
     p=10

label12:
     Write "Abcd"
label16:
 Function Completed
```

Test Case 8

Input:

```
int main()
{
     int a;

     a=input();

     if(a>10)
     {
          while(a>0 && a<10)
          {
               output(a);

               int t=a%10+8-9/7*3&5|9^5+2;

               a-=1;
               a*=10;
               a/=10;
               a-=1;
          }
     }

     else
     {
          switch(a)
          {
               case 1:
                    while(b>5)
                    {
                         output("ABCD");
                         b-=4;
                    }

                    if(a>10)
                    {
                         output(n);
                    }

                    break;

               case 2:
                    output("Hello World\n");

               case 3:

                    int k=10;
                    int t;

                    t=k*10+6;
          }
     }
}
%%
```

Output:

```
     Define function main

label58:

label2:
     a=input

label4:
     t1=a>10
     If t1 is false go to  label54

label21:

label19:
     t2=a>0
     t3=a<10
     t4=t2&&t3
     If t4 is false go to  label20

label17:

label5:
     Write a

label6:
     t5 =a%10
     t6 =t5+8
     t7 =9/7
     t8 =t7*3
     t9 =t6-t8
     t10 =t9&5
     t11 =t10|9
     t12 =5+2
     t13 =t11^t12
     t=t13

label8:
     t14 =a-1
     a=t14

label10:
     t15 =a*10
     a=t15

label12:
     t16 =a/10
     a=t16
```

```
label14:
     t17 =a-1
     a=t17
     Return to  label19
label20:
     Goto  label57

label54:

label53:

label51:
     If a!=1goto label49

label29:
     t18=b>5
     If t18 is false go to  label30

label27:

label23:
     Write "ABCD"

label24:
     t19 =b-4
     b=t19
     Return to  label29
label30:

label30:
     t20=a>10
     If t20 is false go to  label36

label33:

label31:
     Write n
label36:
label52:
     Goto label48

label49:
     If a!=2goto label47

label37:
     Write "Hello World\n"
label50:

label47:
     If a!=3goto label48

label39:
     k=10
```

```
label42:
     t21 =k*10
     t22 =t21+6
     t=t22
label48:
     Goto  label57
label57:
label59:
 Function Completed
```

Assumptions & Limitations

- The sub-C language does not support if else construct without curly braces enclosing the statements that follow.
- This language does not support for and do-while loop constructs.
- This language does not support strings or character arrays.
- Support for constant variables or characters as labels of switch cases are also not supported.
- Default clause is absent in switch statements.
- Declaration of multiple variables in the same line is not supported.
- Else if construct is not supported.
- Void type for functions is not supported. A function with any return type which does not explicitly return a value is considered as void and if used in any assignment, a garbage value is returned.
- Continue statements are not supported.
- The programs are assumed to end with “%%”

Bibliography

References:

- **“Compilers- Principles, Techniques and Tool”- Alfred V Aho, Monica S Lam, Ravi Sethi, Jeffrey D Ullman**
- **“Programming Language Concepts”- Carlo Ghazzi, Jayazeri**
- **“Flex and Bison”- John Levine**