AST.h

```
typedef enum {NB=0, _IDF = 1, BOOLEAN = 2, OP=3}Type_Exp
typedef enum {Int, Bool, Double} Type;
typedef enum {plus, moins, mult, _div} Type_Op;
typedef enum {false, true} boolean;
struct Exp ; typedef struct Exp * AST;
typedef union {double nombre ;char *idf;boolean bool;
 struct {Type_Op top;AST expression_gauche ;
         AST expression_droite ; } op;
             } ExpValueTypeNode;
typedef struct Exp {
  Type_Exp typeexp ;Type typename;
  ExpValueTypeNode noeud ;
                    }expvalueType;
```

AST.C

```
•  AST arbre_gauche(AST a){return a->noeud.op.expressi
•  Type_Op top(AST a){return a->noeud.op.top;}
•  Type  type(AST a){return a->typename;}
•  boolean est_feuille(AST a){return(a->typeexp != OP);
•  AST creer_feuille_booleen(boolean b){AST result
  result->typeexp=BOOLEAN;result->noeud.bool = b;
  result->typename = Bool;return result;}
```

- **Ast acrrer_feuille_idf(char *idf, Type type){**

```
Ast resultat=(ast)malloc(sizeof(struct exp));
Resultat->typeex=idf; resultat->typename =type;
resultat->noeud.idf=
(char*)malloc(sizeof(char)*strlen(idf)+1;
strcpy(result->noeud.idf,idf);
return result;}
```

CFG.h

```
typedef enum {PrintIdf, PrintString, AssignArith, AssignBool,
IfThenArith, IfThenElseArith, For} Type_INST ;
typedef struct INST {Type_INST typeinst;
 union  {// PRINT idftoprint
   struct  {int rangvar; // indice de l'idf} printnode;
   // left := right
   Struct {int rangvar; AST right;} assignnode;
   // IF ... THEN
   struct  {
     int rangvar; AST right;struct LIST_INST * thenlinst;
     struct LIST_INST * elselinst;} ifnode;
     // for (index:=min max) loop list_inst end loop;
   struct {int rangvar;int borneinf; int bornesup;
struct LIST_INST * forbodylinst;} fornode;
***************//SWITCH
Struct {int ragvar;//int nbcase;struct case *cases;struct list_int
*listinstdefault;}switchnode
} node;} instvalueType;

Typedef struct case{int value;struct list int
caasebody;}casevalueinst;
typedef union {
  varvalueType varattribute; constvalueType constattribute; Type
typename;       instvalueType  instattribute;
listinstvalueType  listinstattribute;  } valueType;
```

CFG.c

```
instvalueType*creer_instruction_print(int
rangvar){instvalueType * printinstattribute =
(instvalueType *) malloc(sizeof(instvalueType));
printinstattribute->typeinst = PrintIdf;printinstattribute-
>node.printnode.rangvar = rangvar;return
printinstattribute;}
instvalueType* creer_instruction_affectation(int
rangvar, AST * past){instvalueType * pinstattribute =
(instvalueType *) malloc (sizeof(instvalueType));
pinstattribute->typeinst =
(type(*past)==Bool)?AssignBool:AssignArith;
pinstattribute>node.assignnode.rangvar = rangvar;
pinstattribute>node.assignnode.right = * past;
return pinstattribute;}
```

```
typedef struct LIST_INST {
  struct INST first;
  struct LIST_INST * next;
} listinstvalueType;
typedef struct { Type typename;  double valinit;      }
constvalueType;
```

```
void interpreter_inst(instvalueType
instattribute){ double rexp;
  switch(instattribute.typeinst){
 case PrintIdf : if
(typevar(instattribute.node.printnode.rangvar) == Bool)
{
   printf("%s\n",
((valinit(instattribute.node.printnode.rangvar)==false)?"false
":"true"));
  }else{ printf("%lf\n",
valinit(instattribute.node.printnode.rangvar));   }break;
case AssignArith
:set_valinit(instattribute.node.assignnode.rangvar,instattribut
e.node.assignnode.rght) ;break ;
caseif:if(valinit(instattribute.node.ifnode.rangvar)==instattrib
ute.node.ifnode.right)
 interpreter_list_inst(attribute.node.ifnode.thenlinst);break;
case for:
for(i=instattribute.node.fornode.rangvar,i<instattribute.node.f
ornode.min,i++);//TS1=2
set_valuint(instattribute.node.fornode.rangvar,i)
interpreter_list_inst(instattribute.node.fornode.forbodylinst);
break:
case switch: int i =0
while(i<instattribute.node.switchnode.nbcases)&&
(valinit(instattribute.node.switchnode.rangvar)!=
instattribute.node.switchnode.cases[i].value) i++;
if(i<instattribute.node.switchnode.nbcases){interpreter
list_inst(instattribute.node.switchnode.cases[i].casebodylinst)
;}
else {
interpreterlist_inst(instattribute.node.switchnode.defaultbody
linst);}break;
void interpreter_list_inst(listinstvalueType *
listinstattribute)
{ if (listinstattribute != NULL)
 interpreter_inst(listinstattribute->first);
 interpreter_list_inst(listinstattribute->next); }
void inserer_inst_en_queue(listinstvalueType *
plistinstattribute, instvalueType instattribute)
   { listinstvalueType * liste = (listinstvalueType *)
malloc(sizeof(listinstvalueType));
 liste->first = instattribute;
 liste->next = NULL;
 if (plistinstattribute->next == NULL) plistinstattribute->next
= liste;
 else listinstvalueType * pliste = plistinstattribute;
 while(pliste->next != NULL)
 pliste = pliste->next;
 pliste->next = liste;
       }
```

```
"tableSymb.h"
 typedef enum {Int, Bool} type;

 typedef struct {
                                   char *name;
                                    int nbdecl;
                                    type typevar;
                                    boolean correct;
                                    int valinit;
                                    int linenumdecl;
                          } varvalueType;
 typedef struct {
                                    type typename;
                                    int valinit;
                           } constvalueType;
 typedef union {
                                    varvalueType varattribute;
                                    constvalueType constattribute ;
                                    type typename;
                                   } valueType;
 #define YYSTYPE valueType


*void afficherTS();
*boolean inTS(char * varname, int* rangvar);
*Précondiction : inTS(newvar.name, &i) == false
* void ajouter_nouvelle_variable_a_TS(varvalueType
newvar);
```

```
#include "error.h"
define NBERRMAX 100
 static int NBERRDECL = 0;
 static int NBERRINST = 0;
 static error * ERDECL[NBERRMAX];
 static error * ERINST[NBERRMAX];
 void afficher_erreur(errorType et, int line, char* name)
{ printf("ligne %d : %s ", line, name);
 switch (et)
{ case NonDeclaredVar: printf("variable non declaree\n"); break;
  case IncompatibleAssignType : printf("incompatible avec la valeur
d'affectation\n"); break;
  case BadlyInitialised:  printf("variable mal initialisee\n"); break;
  case AlreadyDeclared:  printf("variable deja declaree\n"); break;
  case IncompatibleCompType: printf("incompatible avec la valeur de
comparaison\n"); break;
 }}
void creer_erreur_instruction(errorType et, int line, char* name)
{ ERINST[NBERRINST++]= creer_erreur(et, line, name);}
 void creer_erreur_declaration(errorType et, int line, char* name)
{ ERDECL[NBERRDECL++]= creer_erreur(et, line, name); }
 error * creer_erreur(errorType et, int line, char* name)
{ error * e = (error*) malloc (sizeof (error) );
 e->name = (char *) malloc (strlen(name));
 strcpy(e->name, name);
 e->linenumdecl = line;
 e->errort = et;              return e; }
```

```
int nombre_erreurs(){
return NBRERRDECL +
NBERRINST; }
```

```
void afficher_erreurs()
{ int idecl = 0;
  int iinst = 0;
    while (idecl < NBERRDECL) {
      afficher_erreur(ERDECL[idecl]->errort,
      ERDECL[idecl]->linenumdecl,
      ERDECL[idecl]->name);
      idecl++;
                                       }
  while (iinst < NBERRINST) {
  afficher_erreur(ERINST[iinst]->errort,
  ERINST[iinst]->linenumdecl,
  ERINST[iinst]->name);
  iinst++;   }}
```

```
#include "tableSymb.h"
 #define NBS 100
 static varvalueType TS[NBS];
 static int NBVAR = 0;
void afficherTS(){
       int i=0;
      for (i=0; i<NBVAR; i++) {
 printf("variable %d = %s, de type %s, initialisee à %s, declaree %d
fois\n", I,TS[i].name , TS[i].typevar==Int?"int":"bool",
(TS[i].typevar==Int?itoa(TS[i].valinit):(TS[i].valinit==true?"true":"fals
e")),
                              TS[i].nbdecl
                   );
                                          }     }
void ajouter_nouvelle_variable_a_TS(varvalueType newvar)
{ TS[NBVAR].nbdecl = newvar.nbdecl;
 TS[NBVAR].name = (char *)malloc(strlen(newvar.name));
 strcpy(TS[NBVAR].name,newvar.name);
 TS[NBVAR].linenumdecl = newvar.linenumdecl;
 TS[NBVAR].correct = newvar.correct;
 TS[NBVAR].typevar = newvar.typevar;
 TS[NBVAR].valinit = newvar.valinit;
NBVAR++; }

 boolean inTS(char * varname, int* rangvar){
 int i =0;
          while ((i < NBVAR) && (strcmp(TS[i].name,varname) != 0))
i++;
          if (i == NBVAR) return false;
          else { *rangvar = i; return true;}}
```

```
instvalueType* creer_instruction_if(int rangvar, AST * past,
listinstvalueType * plistthen, listinstvalueType * plistelse){
instvalueType * pinstattribute = (instvalueType *) malloc
(sizeof(instvalueType));
pinstattribute->typeinst = ((plistelse !=
NULL)?IfThenElseArith:IfThenArith);
pinstattribute->node.ifnode.rangvar = rangvar;
pinstattribute->node.ifnode.right = * past;
pinstattribute->node.ifnode.thenlinst = plistthen;
pinstattribute->node.ifnode.elselinst = plistelse; return pinstattribute;}
instvalueType* creer_instruction_for(int rangvar, int borneinf, int
bornesup, listinstvalueType *pplistfor){
instvalueType * pinstattribute = (instvalueType *) malloc
(sizeof(instvalueType));
pinstattribute->typeinst = For;
pinstattribute->node.fornode.rangvar = rangvar;
pinstattribute->node.fornode.borneinf = borneinf;
pinstattribute->node.fornode.bornesup = bornesup;
pinstattribute->node.fornode.forbodylinst = pplistfor;return pinstattribute;}
```

```
pseudocode generer_pseudo_code_ast(AST as t){
pseudocode  pc = (pseudocode)malloc(sizeof (struct pseudocodenode));
 pseudocode valg, vald,pcswap;
switch(ast->typeexp) {case NB :
   pc->first.codop = PUSH;
   pc->first.param._const = ast->noeud.nombr;   break;
case _IDF :
   pc->first.codop = LOAD;
   pc->first.param.var = ast->noeud.idf;
   break;
 case OP :
   valg = generer_pseudo_code_ast(arbre_gauche(ast));
   vald = generer_pseudo_code_ast(arbre_droit(ast));
 switch(ast->noeud.op.top){
   case plus:   pc->first.codop = ADD; break;
    pc->next = NULL;
case moins :   pc->first.codop = SUB; // opération non commutative
    pc->next = NULL;
    pcswap = (pseudocode)malloc(sizeof (struct pseudocodenode));
    pcswap->first.codop = SWAP;
    pcswap->next = pc;
    pc = pcswap;
    break;
   case mult : pc->first.codop = _MULT;
    pc->next = NULL;
    break;
   case _div:     pc->first.codop = _DI ;
 pc->next = NULL;
    pcswap = (pseudocode)malloc(sizeof (struct pseudocodenode));
    pcswap->first.codop = SWAP;  pcswap->next = pc;
    pc = pcswap;   break ;  }
   inserer_code_en_queue(vald, pc);
   inserer_code_en_queue(valg, vald); pc = valg;
   break; }return pc;}
```

```
ypedef enum { NonDeclaredVar,
                    BadlyInitialised,
                    AlreadyDeclared,
                   IncompatibleAssignType,
                    IncompatibleCompType} errorType;
 typedef struct {
                    char *name; // nom de l'identificateur
qui pose problème
                    int linenumdecl;_ errorType errort;
                   } error;  ERROR.H
```

```
double evaluer(AST ast)
{ double valg, vald;
switch(ast->type) {
 case NB : return ast->noeud.nombre;
break;
 case IDF : return value(ast->noeud.idf);
break;
 case OP :   valg =
evaluer(arbre_gauche(ast));
 vald = evaluer(arbre_droit(ast));
 switch(ast->noeud.op.top)
{ case plus : return valg + vald; break;
 case moins : return valg - vald; break;
 case mult : return valg * vald; break; }
 break; } }
```

```
AST creer_noeud_operation(char op, AST arbre_g, AST
arbre_d, Type type){
 if (debug) printf("creer_noeud_operation()\n");

 AST result= (AST) malloc (sizeof(struct Exp));
 result->typeexp=OP;
 result->typename = type;
 result->noeud.op.top = ((op=='+')?plus:((op=='-
')?moins:((op=='*')?mult:_div)));
 result->noeud.op.expression_gauche = arbre_g;
 result->noeud.op.expression_droite = arbre_d;

 if (debug) printf("out of creer_noeud_operation()\n");

 return result;
}
```

```
void afficher_postfixe_arbre (AST ast){
  // if (est_feuille(ast)){
  switch(ast->typeexp) {
  case BOOLEAN : printf(" %s",(ast-
>noeud.bool==true)?"true":"false"); break;
  case NB : printf(" %lf",ast->noeud.nombre);
break;
  case _IDF :  printf(" %s",ast->noeud.idf);
break;
  case OP :
    afficher_postfixe_arbre(arbre_gauche(ast));
    afficher_postfixe_arbre(arbre_droit(ast));
         switch(ast->noeud.op.top){idem infixe
```

```
void afficher_infixe_arbre (AST ast){
  // if (est_feuille(ast)){
  switch(ast->typeexp) {
  case BOOLEAN : printf(" %s",(ast-
>noeud.bool==true)?"true":"false"); break;
  case NB : printf(" %lf",ast->noeud.nombre); break;
  case _IDF :  printf(" %s",ast->noeud.idf); break;
  case OP :
    printf("gauche [ ");
                    afficher_infixe_arbre(arbre_gauche(ast
)); printf(" ]");
    switch(ast->noeud.op.top){
    case plus : printf(" + "); break;
    case moins : printf(" - "); break;
    case mult : printf(" * "); break;
    case _div : printf(" / "); break;
    }
        printf("droit [ ");
afficher_infixe_arbre(arbre_droit(ast)); printf(" ]");
    break;
  }
}
```

Tete de lecture du fichier : yylex
Tete de lecture des regles : S

Automote mono etat :e pas à pas T mais T*

```
void inintialiser_machine_abstraite(){VM_STACK = creer_pile();}
void interpreter_pseudo_instruction(struct pseudoinstruction pi,
char ** next_label_name){ Element op1, op2,
 resultat; int* rangvar = (int*) malloc(sizeof(int));
 *next_label_name = NULL;
 switch(pi.codop){
case DATA:varvalueType nv  nv.name = (char*) malloc(sizeof(char) *
strlen(pi.param.nv.name)+1);
strcpy(nv.name, pi.param.nv.name);
nv.valinit = pi.param.nv.value;ajouter_nouvelle_variable_a_TS(nv);
                    break;
  case ADD:op1 = depiler(VM_STACK); op2 = depiler(VM_STACK);
resultat = op1 + op2;
 empiler(VM_STACK, resultat); break;
case _DIV:op1 = depiler(VM_STACK); op2 = depiler(VM_STACK);
resultat = op1 / op2;
empiler(VM_STACK, resultat; break;
 case _MULT:  op1 = depiler(VM_STACK); op2 =
depiler(VM_STACK); resultat = op1 * op2;
 empiler(VM_STACK, resultat);break;
case SUB:op1 = depiler(VM_STACK); op2 = depiler(VM_STACK);
resultat = op1 - op2;
empiler(VM_STACK, resultat);break;
 case LOAD:  if(inTS(pi.param.var,rangvar)=true)
empiler(VM_STACK, valinit(*rangvar));break;
 case STORE:op1 = depiler(VM_STACK);
 inTS(pi.param.var, rangvar); set_valinit(*rangvar, op1); break;
case DUPL:    op1 = depiler(VM_STACK); empiler(VM_STACK, op1);
 empiler(VM_STACK, op1); break;
case PUSH:empiler(VM_STACK, pi.param._const); break;
case SWAP: op1 = depiler(VM_STACK); op2 = depiler(VM_STACK);
empiler(VM_STACK, op1);
 empiler(VM_STACK, op2); break;
 case JNE:op1 = depiler(VM_STACK); op2 = depiler(VM_STACK);
if (op1 != op2) {*next_label_name = (char*) malloc
(strlen(pi.param.label_name)+1);
 strcpy(*next_label_name, pi.param.label_name);}else {;}break;
 case JG: op1 = depiler(VM_STACK); op2 = depiler(VM_STACK);
if (op1 > op2) {*next_label_name = (char*) malloc
(strlen(pi.param.label_name)+1);
strcpy(*next_label_name, pi.param.label_name);}else {;break;
case JMP:*next_label_name = (char*) malloc
(strlen(pi.param.label_name)+1);
 strcpy(*next_label_name, pi.param.label_name);break;
case PRNT: op1 = depiler(VM_STACK); printf("%lf", op1); break;
case LABEL:  break;}}
void interpreter pseudo_code_list_inst(pseudocode pc)
{if(pc !=NULL)    {interpreter pseudo_code_inst()pc->first ;  interpreter
psudi_code_list_inst(pc->next ;)
```

```
void interpreter_pseudo_code(pseudocode pc){
char ** next_label_name= (char**)malloc(sizeof(char*));
if(pc!=NULL) {interpreter_pseudo_instruction(pc-
>first,next_label_name);
 if(*next_label_name==NULL)
interpreter_pseudo_code(pc->next);//pas de branchment
else{//JNE JMP effectuer un branchment (o(n))
 struct pseudocodenode *compteur_ordinal=pc->next ;
 while((compteur_ordinal->first.codop !=LABEL)||
 (strcmp(compteur_ordinal->first.param.label_name,*
next_label_name)!=0)}
 { compteur_ordinal = compteur_ordinal->next;}
 interpreter_pseudo_code(compteur_ordina->nextl)
```

```
void afficher_pseudo_instruction(struct pseudoinstruction pi){
 switch(pi.codop){
 case ADD: printf("ADD\n"); break;
 case DIV: printf("DIV\n"); break;
 case DUPL: printf("DUPL\n"); break;
 case LABEL: printf("%s:\n",pi.param.label_name); break;
 case LOAD: printf("LOAD "); printf("%s\n",pi.param.var); break;
 case MULT: printf("MULT\n"); break;
 case POP: printf("POP\n"); break;
 case PUSH: printf("PUSH "); printf("%lf\n",pi.param._const);
break;
 case SUB: printf("SUB\n"); break;
 case STORE: printf("STORE "); printf("%s\n",pi.param.var);
break;
 case SWAP: printf("SWAP\n"); break;
 case PRNT: printf("PRINT\n");break;
 case JNE: printf("JNE ");
printf("%s\n",pi.param.label_name);break;
 case JMP: printf("JMP ");
printf("%s\n",pi.param.label_name);break;
 }}
```

```
pseudocode generer_pseudo_code_list_inst(listinstvalueType *
plistinstattribute){pseudocode  pc1=NULL, pc2=NULL;if (plistinstattribute != NULL){
 pc1 = generer_pseudo_code_inst(plistinstattribute->first);
 pc2 = generer_pseudo_code_list_inst(plistinstattribute->next);
  inserer_code_en_queue(pc1, pc2);}return pc1
```

Rec gauche : econome
Rec droite : gourmande
En VFN , evaluation G->A :rec gauche et de D à D :rec droite

LL1 analyseur descendant genere automate(à pile)
Rec à gauche peut produite le bouclage de l'analyseur top_down car elle met indifiniment l'arbre sans generer le symbole teermninal
Rec droite : adapté au parser top-down

LL1 : determinisme
Ambiguité :semi decidable
Non **ambiguite** : non decisable
Ambiguité : non determinisme

RI : la production de RI graphyqie pas necessaire , en effet l'analyseur syntaxico semantique suit un sens (descendant , ascendant)ce qui limite le sens de calcul des attribut qui peuvent etre soit synthe ou herité,il est pplus pratique de stocker le resultat de lanaly syntaxico semantique pr effectuer des calculs dattributs necessitant parcours ascendant ou descendant sans se lier au sens de lanalyse semantic syntaxique .

Pseudo code :tourne sur une machine virtuelle
Code :tourne sur machine hysique
Mémoire virtuelle : mémoire code : pr changmnt de code(sous forme de liste)
CFG jour le role de la mémoire liste
Mémoire statiqu :pr allocation dynamique(decl var globale) (table de symbol pr zz)
Tas(heap) : pr allocation dyanique (zz pas de heap)
Pile (stack) : pr variable locale,variable de retour , val de parametre
Toute machine admet une pile , elle peut ne as avoir de registre
*__code 1adress__e : tt ds la pile
Stack machine code : 1 adre
***code 2adress** :Archi CISC
***code 3adrs** :rchi RISC
***java** ; RI linaeaire
***EXP arithmetique** : necassaire prg en pseudo code
***grammaire nullable** :analogie vec eps NFA
***LAG** :langage binaire
***optimisation en mémoire** : Minimiser un automate
***Automa detat fini** : langage regulier
***erreur parenthesé** : analy syntaxique
***recursivité gauche** : bouclage parseur LL1
***grammaire reguliaire** :grammaire lineraire
DAG : representation intermediare graphique
***automa à pile** !langage irregulier
Automate detat fini :regulier
***code a 2 adrs** :RI lineaire
*__anb__n :suit cfg lineaire n<42ouss 51 -1
***grammaire ambigue** : analyse floue
Stack machine code : gen code à1 adress
LALR : LR : analyseur descendant top down
Gram heriditairemnt ambigue : analyse impo
Gram algebriqu gram linaire
Derivation gauche et droite : tri et tri inverse
Commentaire /*sans/* : anal lexical
Grammaire nn ll : alanyseur descndant non optimal
1 adresse code est choisi pr sa taille de code
Automate NFA : graimmeure ambihue
Grammaier attribué : action sémantique
3 adress : rapidité
Embarqué : code optimisé
Terminal t : classe regulière de sigma*
Role representation inter : resolution surcharge (oui) , factoristion d certaine optimisation, decompostition en plusieurs etapes de traduction , independance des partie frontale et terminale (non)
Byte codejava : one adresse code
Acorn risq machine arm : 3adrs
Type void * vector : fermeture de klleeen
Addop rg1reg2 :2 adre

```
void  afficher_pseudo_code(pseudocode  pc){
 if (pc != NULL){
afficher_pseudo_instruction(pc->first);
afficher_pseudo_code(pc->next); }}
void inserer_code_en_queue(pseudocode  pc1,
pseudocode pc2){
  if (debug) {
    afficher_pseudo_code(pc1);
    afficher_pseudo_code(pc2);
  }
  if (pc1->next == NULL) {
    pc1->next = pc2;
  }else{
    pseudocode pc = pc1;
    while(pc->next != NULL) {
      pc = pc->next;
    } pc->next = pc2;}
  if (debug) {
afficher_pseudo_code(pc1);}}
```

interpreteur pseudo code : branchement arriere a des label se trouvant avec l'inst JMP n'est as possible
Effectuer un brachement s'effectue en cout de la boucle au ire des cas en O(n) et ne peut être optimisé par une table de hachage de O(1).

RI graphique :DAG,AST,arbre a
RI lineaire : Bytecode,tree adress code
RI hybride :: CFG, control flow graphe

```
typedef enum {ADD, DIV, DUPL, JMP, JNE, LABEL, LOAD,
MULT,
POP, PRNT, PUSH, SUB, STORE, SWAP} CODOP;
 typedef union {char * var;
                    double _const;
                    char * label_name;  struct namevalue nv; } Param;
struct namevalue {char * name;double value;};
 struct pseudoinstruction{ CODOP codop; Param param; };
struct pseudocodenode{ struct pseudoinstruction first;struct
pseudocodenode * nextt};
typedef struct pseudocodenode * pseudocode;
pseudocode
generer_pseudo_code_inst(instvalueType
instattribute){
 static label_index = 0;
pseudocode pc = (pseudocode)malloc(sizeof (struct
pseudocodenode)), pc1,pc2,pc3,pc4,pc5;
 pseudocode rexpcode;
 char * label_name;
 char *label_num;
 switch(instattribute.typeinst){
 // PRINT IDF
 case PrintIdf :
pc->first.codop = LOAD;
 pc->first.param.var = name(instattribute.node.printnode.rangvar);
 rexpcode = (pseudocode)malloc(sizeof (struct pseudocodenode));
 rexpcode->first.codop = PRNT;
 rexpcode->next = NULL;
 pc->next = rexpcode;
 break;
 // IDF ASSIGN EXP
 case AssignArith :
 rexpcode =
generer_pseudo_code_ast(instattribute.node.assignnode.right);
pc->first.codop = STORE;
 pc->first.param.var = name(instattribute.node.assignnode.rangvar);
 pc->next = NULL;
 inserer_code_en_queue(rexpcode, pc);
 pc = rexpcode;                    break;
/ IDF ASSIGN TRUEFALSE
 case AssignBool :
 pc->first.codop = PUSH;
 pc->first.param._const = instattribute.node.assignnode.right0;
 pc1 = (pseudocode)malloc(sizeof (struct pseudocodenode));
 pc1->first.codop = STORE;
 pc1->first.param.var =
name(instattribute.node.assignnode.rangvar);
 pc1->next = NULL;
 pc->next = pc1;                   break;
// IF PARO IDF EGAL EXP PARF THEN LISTE_INST
ENDIF
 case IfThenArith :
 pc = generer_pseudo_code_ast(instattribute.node.ifnode.right);
pc1 = (pseudocode)malloc(sizeof (struct pseudocodenode));
 pc1->first.codop = LOAD;
 pc1->first.param.var = name(instattribute.node.ifnode.rangvar);
 pc1->next = NULL;
 pc2 = (pseudocode)malloc(sizeof (struct pseudocodenode));
 pc2->first.codop = JNE;
 label_num=itoa(label_index++);
 pc2->first.param.label_name = (char*)
malloc(6+strlen(label_num));
 strcpy( pc2->first.param.label_name, "endif");
 strcat( pc2->first.param.label_name, label_num);
 pc2->next = NULL;
 pc3 =
generer_pseudo_code_list_inst(instattribute.node.ifnode.thenlinst );
 pc4 = (pseudocode)malloc(sizeof (struct pseudocodenode));
 pc4->first.codop = LABEL;
 pc4->first.param.label_name = pc2->first.param.label_name;
 pc4->next = NULL;
 inserer_code_en_queue(pc3, pc4); pc2->next = pc3; pc1->next =
pc2; inserer_code_en_queue(pc, pc1); break;
// IF PARO IDF EGAL EXP PARF THEN LISTE_INST ELSE
LISTE_INST ENDIF
 case IfThenElseArith :
 pc = generer_pseudo_code_ast(instattribute.node.ifnode.right);
 pc1 = (pseudocode)malloc(sizeof (struct pseudocodenode));
 pc1->first.codop = LOAD;
 pc1->first.param.var = name(instattribute.node.ifnode.rangvar);
 pc1->next = NULL;
 pc2 = (pseudocode)malloc(sizeof (struct pseudocodenode));
 pc2->first.codop = JNE;
 label_num=itoa(label_index++);
 pc2->first.param.label_name = (char*)
malloc(6+strlen(label_num));
 strcpy( pc2->first.param.label_name, "else");
 strcat( pc2->first.param.label_name, label_num);
 pc2->next = NULL;
 pc3 =
generer_pseudo_code_list_inst(instattribute.node.ifnode.thenlinst );
 pc31 = (pseudocode)malloc(sizeof (struct pseudocodenode));
 pc31->first.codop = JMP;
 pc31->first.param.label_name = (char*)
malloc(6+strlen(label_num));
 strcpy( pc31->first.param.label_name, "endif");
 strcat( pc31->first.param.label_name, label_num);
 pc4 = (pseudocode)malloc(sizeof (struct pseudocodenode));
 pc4->first.codop = LABEL;
 pc4->first.param.label_name = pc2->first.param.label_name;
 pc4->next = NULL;
 pc31->next = pc4;
c5 =
generer_pseudo_code_list_inst(instattribute.node.ifnode.elselinst );
 pc4->next = pc5;
 pc6 = (pseudocode)malloc(sizeof (struct pseudocodenode));
 pc6->first.codop = LABEL;
 pc6->first.param.label_name = (char*) malloc(strlen( pc31-
first.param.label_name)+1);
 strcpy( pc6->first.param.label_name, pc31-
>first.param.label_name);
 pc6->next = NULL;
 inserer_code_en_queue(pc5, pc6);
 inserer_code_en_queue(pc3, pc31);
 pc2->next = pc3;
 pc1->next = pc2;
 inserer_code_en_queue(pc, pc1); break;
 }
 return pc;}
```