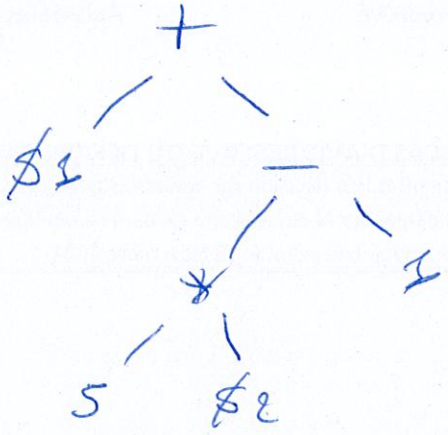
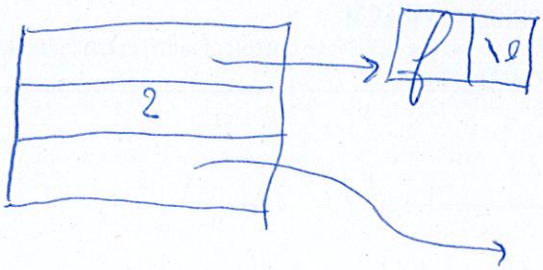
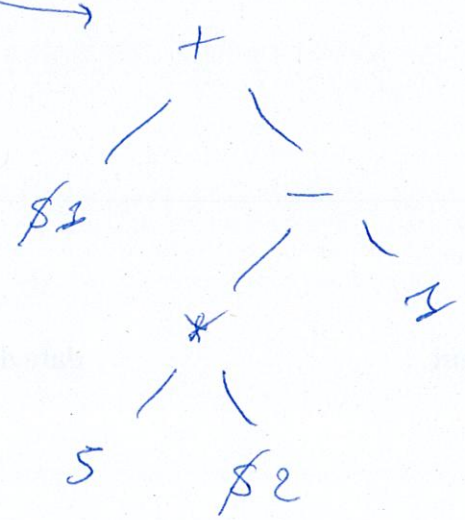
Q3. AST de $1 + 5 * $2 - 1



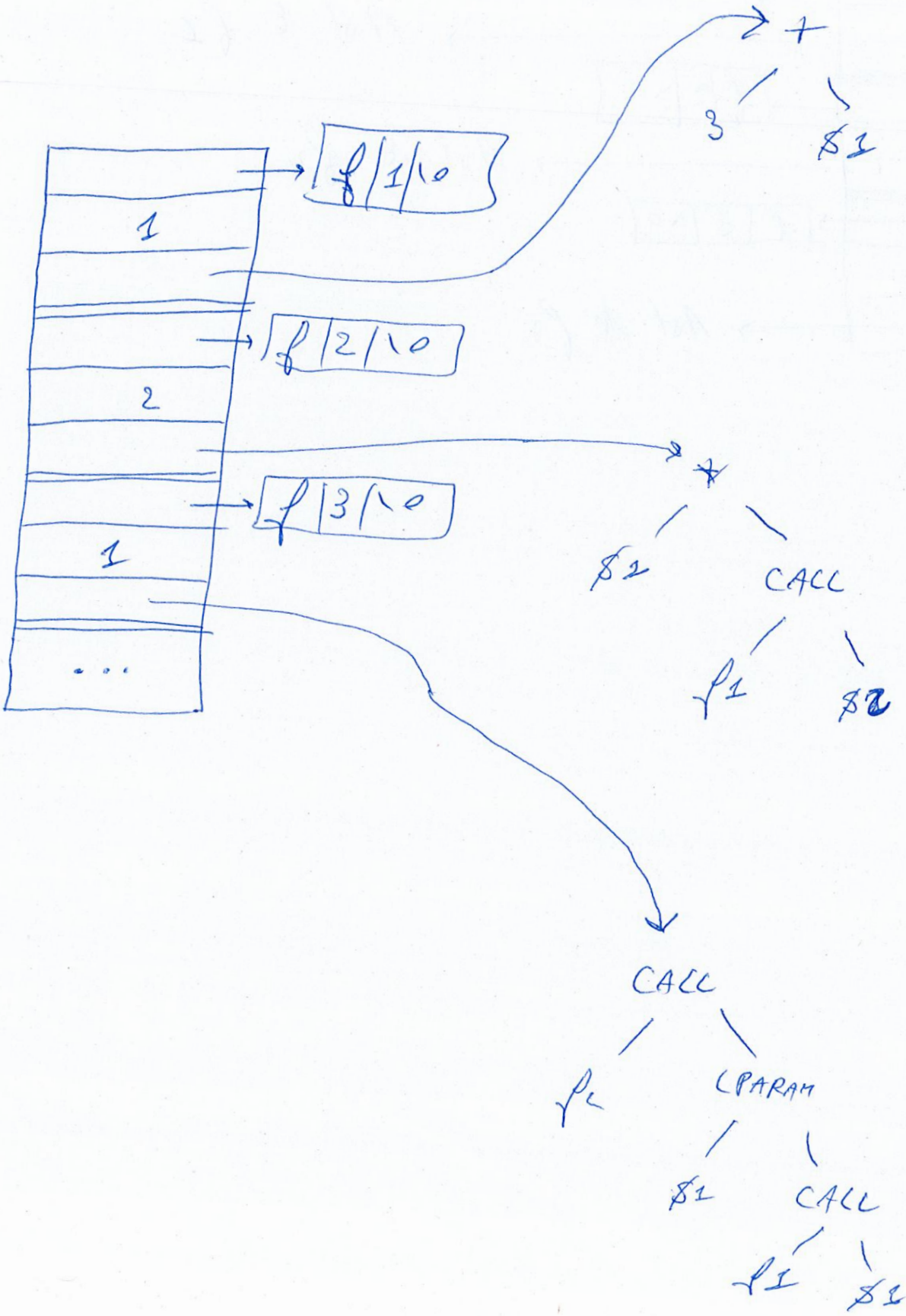
Q4.1 contenu de la structure FoncDec pour { f 2 = $1 + 5 * $2 - 1 ;





Q12 -

Contenu de SeqFoncDec pour f1



Q1. Exemple d'erreur lexicale : #

Q2. Exemple d'erreur syntaxique lexicale : 5 ++ 3

Q3. Voir fichier séparé

Q4.

2. contenu du fichier fonction.h

```
typedef struct {
            char[256] nf ; // nom de la fonction
            int nbp ; // nombre de parametre
        Ast *corps ; // corps
} FoncDec ;

FoncDec creerFd (char *nf, int nbp, Ast *corps) ;
/* cree une structure FoncDec */
```

Q5.

```
void Rec_DecFonc(FoncDec *df) {
ACCO FUNC ENTIER EGAL Eag ACCF
  // on suppose aue l'analyse lexicale a déjà été démarrée ...
  if (LC.nature == ACCO) {
    Avancer() ;
    if (LC.nature == FUNC) {
      strcpy(&(df->nf), LC.chaine) ;  // nom de la fonction
      Avancer() ;
         if (LC.nature == ENTIER) {
             &(df->nbp) = LC.valeur ;  // nbre de paramètres
             Avancer() ;
             if (LC.nature == EGAL) {
               Avancer() ;
             Rec_Eag(&(df->corps))  // corps de la fonction
               if (LC.nature == ACCF)
                  Avancer() ;
               else
                 Erreur() ;
             } else
               Erreur() ;
         } else
           Erreur() ;
    } else
      Erreur() ;
  } else
     Erreur() ;
}
```

Q6.

Pour verifier que les paramètres utilisés dans le corps d'une fonction correspondent bien aux paramètres déclarés, il faut vérifier que, si n est le nombre de paramètres déclarés, alors l'ensmble des paramètres présents dans le corps de cette fonction est {$1, $2, ... $n}.

```
void creerEnsParam (int ensParam, Ast A) {
// cree l'ensemble des parametres presents dans l'Ast A
      switch (a->nature) {
            case N_PARAM:
            // le parametre de numéro noparam est
                  // présent dans A
```

```
                ensParam[A->nparam]=1 ;
                break ;
            case N_PLUS:
            case N_MOINS:
            case N_MULT:
                creerEnsParam (ensParam, A.fg);
                creerEnsParam (ensParam, A.fd);
                break ;
            case N_ENTIER:
                break ;

    }
}

int verifParam (FoncDec df) {
  int ok=0 ;  // valeur de retour
  int nbpd = df.nbp // nombre de paramètres déclarés

     // ensemble de parametres presents dans le corps
     // initialement vide
  int *ensParam = (char *)malloc(nbpd * sizeof(int)) ;
  for (int i=0 ; i<nbpd ; i++)
      ensParam[i] = 0 ;

  creerEnsParam (ensParam, df.corps)

  // on verifie que les nbpd params sont bien présents dans l'ensemble
  for (int i=0 ; i<nbpd ; i++)
      ok = 0k && nbpd[i] == 1 ;

  return ok
}
```

Q11.

* ajout au contenu du fichier fonction.h

```
typedef struct {
            FoncDec tab[256] ; // tableau de declaration de fonctions
            int nbf ; // nombre de fonctions déclarées
} SeqFoncDec ;


void initSFD (SeqFoncDec *sfd) ;
/* cree une structure SeqFoncDec sfd vide */

void insSFD (SeqFoncDec *sfd, FoncDec df) ;
/* ajoute df a sfd */

int rechSFD (SeqFoncDec sfd, char *s) ;
/* renvoie l'indice de la fonction de nom s dans SeqFoncDec
   (-1 si cette fonction n'est pas présente)
*/
```

* ajout au contenu du fonction ast.h

On étend le type TypeAst avec deux nouveaux noeuds :

- N_CALL : appel de fonction,
  son fils gauche est le nom de la fonction appelée,
  son fils droit est la liste des parametres effectifs
- -N_LPARAM : liste de paramètres effctifs
  son fils gauche est un paramètre effectif (une Eag)
  son fils droit est une liste de paramètres effectifs

Q13.

```
// parcours de l'Ast représntant le corps d'une fonction
int execAstAppel(EnsFoncDec LFonc, Ast A, SeqParam params) {
    SeqParam paramcall ; // parametres pour un appel de fonction interne

      switch (a->nature) {
            case N_PARAM:
            return params[A->nparam] ; // renvoie la valeur du parametre
            case N_PLUS:
                  return execAstAppel(EnsFoncDec LFonc, Ast A->gauche, SeqParam
params)
                  +
                  return execAstAppel(EnsFoncDec LFonc, Ast A->droit, SeqParam
params) ;
            case N_MOINS:
                  return execAstAppel(EnsFoncDec LFonc, Ast A->gauche, SeqParam
params)
                  -
                  return execAstAppel(EnsFoncDec LFonc, Ast A->droit, SeqParam
params) ;
            case N_MULT:
                  return execAstAppel(EnsFoncDec LFonc, Ast A->gauche, SeqParam
params)
                  *
                  return execAstAppel(EnsFoncDec LFonc, Ast A->droit, SeqParam
params) ;
                  break ;
            case N_ENTIER:
                  return a->val ;
            cas N_CALL:
                  // retrouver les parametres
                  paramcall = getParams(A->droit) ;
            return execAppel (LFonc, rechSFD(LFonc, A->gauche, paramcall) ;
    }
}

int execAppel(EnsFoncDec LFonc, FoncDec df, SeqParam parame) {
      Ast A=df.corps ; // corps de la fonction a exécuter
    return execAstAppel(LFonc, A, parame) ;
}
```

Q13.

```
void Rec_ListeParamE (SeqParam *parame) {
   Ast A ; // Ast d'un parametre effectif
   initSeqParam(parame) // initialise parame à vide
   if ((LC.nature() == PARO) || (LC.nature() == ENTIER) ||
       (LC.nature() == PARAM) ||  (LC.nature() == IDF)) {
            Rec_Eag (&A) ;
        evaluer(A) ; // calcule la valeur du parametre effectif courant
        insParem (v, parame) ; // ajoute un nouveau parametre
   } else {
      // pas de paramètres effectifs
   }
}


void Rec_CallFonc (FonDec *df) {
   if (LC.nature() == IDF) {
```

```
       Avancer()
     if (LC.nature() == PARO) {
          Avancer()
        Rec_ListeParamE (SeqParam &parame) ;
        if (LC.nature() == PARF) {
             Avancer()
          } else Erreur() ;
       } else Erreur() ;
   } else Erreur() ;
}

void Rec_ListeDecFonc (SeqFonDec *LFonc) {
      FoncDec df ;
   initSFD (Lfonc) ; // initialise LFonc à vide
   if (LC.nature() == ACCO) {
      Rec_DecFonc(&df) ;
      Rec_ListeDecFonc(Lfonc) ;
    insSFD (Lfonc, df) ; // on construit la liste globale
   } else {
      // cas epsilon, pas de fonctions auxilliaires
   } ;
}

int executerPgm() {
      SeqFoncDef LFonc;  // les fonctions auxilliaires
    FoncDef df ;  // nom de la fonction principale
    SeqParam parame ; // parametres effectifs de la fonction principale
      Rec_ListeDecFonc(&Lfonc) ;
    Rec_CallFonc(&df, &parame) ;
    if (LC().nature == FSEQ)
         return execAppel(LFonc, df, params) :
    else
            Erreur() ;
      return 0 ;
    }
}
```