Table des matières

[1. Les parties réaliser et non réaliser : 3](#_Toc51086119)

[1.1. Les parties réaliser : 3](#_Toc51086120)

[1.2. Procédure à suivre pour lancer le projet : 3](#_Toc51086121)

[2. Les fonctions : 3](#_Toc51086122)

[2.1. Partie Sémantique : 3](#_Toc51086123)

[2.3. Partie de génération du code assembleur : 4](#_Toc51086124)



Rapport du Projet compile

Trinôme :

Nom : SAGGOU

Prénom : OMAR ACHREF EDDINE

Matricule : 201500008961

-------------------------------------------------

Nom : OUBOUABDELLAH

Prénom : Mohamed

Matricule : 201500009643

-------------------------------

Nom : BENTAMA

Prénom : Oussama Amine

Matricule : 201500008546

## Les parties réaliser et non réaliser :

### Les parties réaliser :

* Analyse lexical.
* Analyse syntaxique avec les routines sémantiques.
* Gestion de la table de symboles.
* Génération du code intermédiaire.
* Traitement des erreurs.

## Procédure à suivre pour lancer le projet :

En utilisant l’inviter de commande ‘**’cmd** ‘’ on exécute la commande ‘’**com.bat**’’.  
Cette commande contient :

* Flex flex.l ( pour construit un analyseur lexical )
* Bison -d bison.y ( pour construit un analyseur syntaxique )
* gcc lex.yy.c bison.tab.c -o flex.exe (pour générer le compilateur)

## Les fonctions :

### Partie Sémantique :

#### Traitement de la table de symbole :

1. **void init();**

Pour initialiser la table de symbole.

1. **int hachage(char chaine[]);**

Pour hacher les variables stocker dans la table de symbole en codant chaque syllabe d’identifiant.

1. **int rechercher(char nom [], element \*\* in);**

Pour la recherche des varibale dans la table desymbole.

1. **void inserer (char nom[], int code,char nature[30],char taille[255]);**

Pour l’insertion des variables avec ses différentes informations dans la table de symbole.

1. **int gettype(char nom[]);**

Pour retourner le type de la variable.

1. **void afficher();**

Pour afficher le contenue de la table de symbole.

1. **void modifier( char nom[], int code);**

Pour changer le type d’une variable en utilisant un code.

1. **char\* getValeur(char\* idf);**

Pour retourner la valeur d’une variable.

1. **void setValeur(char\* idf,char\* valeur);**

Pour la mise à jour de la valeur d’une variable.

1. **char\* typeVar(int i);**

Pour retourner le type correspondant d’après le code numérique.

1. **void err\_taille\_tab(int type,char val[]);**

Pour informer le dépassement de taille des variables de type tableau.

1. **void err\_incompa\_typ\_tab(int typtab,int typvar);**

Pour informer l’incompatibilité des variables de type tableau.

1. **void err\_incompa\_typ\_var(int typdec,int typvar);**

Pour informer l’incompatibilité des variables.

#### Partie code intermédiaire :

1. **void generer(char\* a, char\* b, char\* c, char\* d);**

Pour insérer les informations dans le quadruplés.

1. **void quadL(int i, char\*b, char\* c, char\* d);**

Pour générer les quadruplés **de comparaison d’égalité et différence** utiliser dans les conditions de :

* **IF**
* **WHILE**
* **FOR**

1. **void quadC(int i, char\* b, char\* c, char\* d);**

Pour générer les quadruplés **de comparaison de supériorité et d’infériorité** utiliser dans les conditions de :

* **IF**
* **WHILE**
* **FOR**

1. **void afficher\_qdr();**

Pour afficher l’intégralité des quadrulés.

1. **void delete\_quad();**

Pour supprimer les quadruplés des variable et variable temporaire non utilisées.

### Partie de génération du code assembleur :

#### Fonction :

**void assembler();**

Pour générer le code assembleur 8086.