Rapport TCP

I. Définition informelle du langage source

Un programme TPC est une suite de fonctions.
Chaque fonction est constituée de déclarations de constantes (à porter global ou local) et variables et d'une suite d'instructions.
Le langage TPC possède deux types : `int` et `char`, un troisième mot clé `void` définie l'absence de valeurs de retour/argument de fonction.
Les mots clef `print` / `readc` et `reade` sont des fonctions fournies par le langage.
Le but premier du langage est de géré des entrées/sorties de texte.

AI. Éléments lexicaux

\+ : addition ou plus unaire
\- : soustraction ou moins
unaire * : multiplication
/ et % : division et reste de la division entière
Pour tous les élément lexicaux précédent opérateur= est reconnu (+=/-= ...) = : opérateur d'affectation
! : négation booléenne
==, !=, <, >, <=, >= : les opérateurs de comparaison
&&, || : les opérateurs booléens
; et , : le point-virgule et la virgule
(,),{, },[et]: les parenthèses, les accolades et les crochets

BI. Mot clé/fonction fourni par le langage TCP

Les mots clef `if`, `else`, `while`, `do`, `for`, `switch`, `case`, `default`, `break`, `void` et `return` on les mêmes utilization que celle défini dans le langage C. `print` permet d'afficher un entier ou un caractère.
`reade(IDENT)` et `readc(IDENT)` permettent respectivement le lire un entier et un caractère saisie au clavier.
Le mot clé break est uniquement reconnu dans le cadre d'un switch.
La syntaxe de `const` est différente de celle du C, elle est utilisée de la manière suivante : `const IDENT = valeur` (syntaxe plus proche d'un #define en C bien qu'elles ont un comportement totalement différent).
Le mot clé value sera développé dans la partie ajout.

IV. Ajout personnel

Le sujet étant très vaste et laissant libre recourt à nôtre imagination en se qui concerne l'ajout de fonctionnalité pour le langage TCP.

Certain ajout basique vis-à-vis de la grammaire ont été très simple à implémenté notamment les tableaux à plusieurs dimensions (quelques mots à modifier dans la grammaire fournie par l'énoncé).
Les commentaires format C++ sont aussi autorisé.
La structure `do{}while()` ; a aussi été très simplement implémenté.
Certain structure plus complexe ont aussi été implémenté, notamment la boucle for, chaque champs du for peut être vide, plusieurs instructions peuvent être misent dans le bloc d'initialisation et d'incrémentation, dans se cas si comme pour le for du C elles seront alors séparées par des virgules.
Pour simplifier la vie du programmateur les opérateur « += » « -= » « /= » »%= » et « *= » on été implémenté.

La structure switch case aussi été implémenté avec une syntaxe identique à celle du C.
Les fonction de prenant aucun paramètre peuvent être défini sans le mot-clé `void` exemple :

```
int fct(){
    // code
}
```

En se qui concerne les messages d'erreur, comme un « syntaxe error, line 4 » nous paraissais un peu léger on a aussi déclaré

%define parse.error verbose

Bien qu'a l'origine on l'utilise pour le débogage, et qu'il n'est pas toujours compréhensible par un l'utilisateur, il permet tout de même de souligné des erreurs de syntaxe tel que l'utilisation d'un caractère non reconnu par le langage (dans se cas si on a ajouté une couche dans le lex qui affiche le caractère en question), l'oubli d'un point-virgule . . .

Comme dit précédemment le but du langage TPC est de gérer des entrées/sorties de texte, une utilisation qui semble naturel est donc appliquer des algorithmes de cryptographie sur des chaînes de caractères (on utilisera la convention du C : une chaîne de caractère est un tableau de char tels que le dernier élément utilise est suivit de \0).
Un algorithme très simple à mêmes en place est le cryptage par substitution qui peuvent être implémenté dans le langage TPC de la manière suivante :

```c
void sub_crypto_switch(char texte[]){
    int i;
    for(i = 0; texte[i] != '\0'; i = i + 1){
        switch(texte[i]){
            case('a'): texte[i] = 'e'; break;
            case('z'): texte[i] = 'j'; break;
            case('e'): texte[i] = 'p'; break;
            case('r'): texte[i] = 'n'; break;
            case('t'): texte[i] = 'y'; break;
            case('y'): texte[i] = 't'; break;
            case('u'): texte[i] = 'f'; break;
            case('i'): texte[i] = 'r'; break;
            case('o'): texte[i] = 'a'; break;
            case('p'): texte[i] = 'm'; break;
            case('q'): texte[i] = 'i'; break;
            case('s'): texte[i] = 's'; break;
            case('d'): texte[i] = 'u'; break;
            case('f'): texte[i] = 'h'; break;
            case('g'): texte[i] = 'l'; break;
            case('h'): texte[i] = 'z'; break;
            case('j'): texte[i] = 'c'; break;
            case('k'): texte[i] = 'o'; break;
            case('l'): texte[i] = 'i'; break;
            case('m'): texte[i] = 'w'; break;
            case('w'): texte[i] = 'k'; break;
            case('x'): texte[i] = 'd'; break;
            case('c'): texte[i] = 'x'; break;
            case('v'): texte[i] = 'b'; break;
            case('b'): texte[i] = 'v'; break;
            case('n'): texte[i] = 'g'; break;
        }
    }
}
```

Sauf que c'est assez bourrin et que dans toutes les case on modifie texte[i]
On a donc pensez que implémenté une structure inspiré du switch dans le but d'alléger la syntaxe pour être une bonne idée.
On s'est premièrement pensez qu'implémenter qu'un structure (que l'on nommera value) similaire au switch mais qui elle, renverra une valeur pour être un bon début (en sois value serait au switch ce que le ?: est au if dans le langage C).
Dans un second temps, une fois de plus dans l'objectif d'alléger la syntaxe on a retiré le mot-clé case qui pourrai très bien être implicite, on a donc obtenu une structure tel que le comportement de la fonction ci-dessus et ci-dessous sois le même :

```
void sub_crypto_value(char texte[]){
    int i;
    for(i = 0; texte[i] != '\0'; i += 1){
        texte[i] = value(texte[i]){
            'a': 'e';
            'z': 'j';
            'e': 'p';
            'r': 'n';
            't': 'y';
            'y': 't';
            'u': 'f';
            'i': 'r';
            'o': 'a';
            'p': 'm';
            'q': 'i';
            's': 's';
            'd': 'u';
            'f': 'h';
            'g': 'l';
            'h': 'z';
            'j': 'c';
            'k': 'o';
            'l': 'i';
            'm': 'w';
            'w': 'k';
            'x': 'd';
            'c': 'x';
            'v': 'b';
            'b': 'v';
            'n': 'g';
            default: texte[i];
        };  // pas oublier le ';'  car c'est une instruction
    }
}
```

Noté que contrairement au switch, value à obligatoirement besoin d'un default.