université de BORDEAUX

COMPILATION

4TIN603U

Rapport de Projet

Jean-Christophe Blin – Gabriel Dubois – Quentin Fergelot - Yohan Lematre

18 mai 2020

```
println "Veuillez vous identifier.";
print "Code secret : ";
readln code;

test = true;

switch(i + j)                                           //***** Switch optimisé + un seul case + sans default
{
    case 0: println("Un seul case");
}

for(i = 0; i < 32; i = i + 1)                           //***** Structure for
{
    switch(i + j)                                       //***** Switch optimisé avec de nombreux cases
    {
        case 0: print("0 ");
        case 1: println("1");
        case 2:
            if(today == FRIDAY)                         //***** Structure if + avec else
            {
                println("Bravo ! Voici donc le résultat de votre couleur en récompense :");
                switch(color)                           //***** Switch standard
                {
                    case YELLOW: print("Vous avez choisi la couleur Jaune ! (valeur : ");
                    case BLUE: print("Vous avez choisi la couleur Bleu ! (valeur : ");
                    case RED: print("Vous avez choisi la couleur Rouge ! (valeur : ");
                    case GREEN: print("Vous avez choisi la couleur Vert ! (valeur : ");
                    default: println("Vous n'avez choisi aucune couleur, tant pis pour vous.");
                }

                switch(color)                           //***** Switch optimisé + sans default
                {
                    case RED: println ("0)");
                    case BLUE: println ("1)");
                    case YELLOW: println ("2)");
                    case GREEN: println ("3)");
```

```
/* switch ((i + j)) */
int _switch_nb_case__49 = 1;
int _switch_test__49 = (i + j) >= _switch_nb_case__49;
static void* const _switch_label_tab__49[] = {
    && switch_label_case_0__49,
    && switch_label_end__49
};
if (_switch_test__49)
    goto *_switch_label_tab__49[_switch_nb_case__49];
goto *_switch_label_tab__49[(i + j)];

_switch_label_case_0__49:{

    /* println "Un seul case" */
    printf("%s\n", "Un seul case");
    goto _switch_label_end__49;
}

_switch_label_end__49:{}

/* for (i = 0 ; (i < 32) ; i = (i + 1)) */
int _for_test__334;

/* i = 0 */
i = 0;

_for_label_loop__334:{
    _for_test__334 = (i < 32);
    if (!_for_test__334)
        goto _for_label_end__334;
    {

        /* switch ((i + j)) */
        int _switch_nb_case__333 = 31;
        int _switch_test__333 = (i + j) >= _switch_nb_case__333;
        static void* const _switch_label_tab__333[] = {
            && switch_label_case_0__333,
            && switch_label_case_1__333,
            && switch_label_case_2__333,
```

1/ Travail demandé

Dans un premier temps, nous avons réalisé le travail demandé sur les structures *if* (avec son complément *else*), *while*, *for* et *switch* en prenant exemple sur la partie de la structure *if* déjà implémenté. Pour produire le code *C* correspondant au programme *Lea* nous avons essayé de nous rapprocher le plus possible d'un code dans l'esprit assembleur, tout en restant cohérent avec le reste du projet déjà réalisé.

Pour les structures de type boucle nous avons donc traduit ces boucles par des tests "*if (…) goto …*". Ainsi les structures *while* et *for* se rapprochent de la structure *if*.

Nous avons également traité la présence ou non d'un "default" dans la structure *switch*, afin d'optimiser celui-ci.

Nous avons réfléchi à la possibilité d'entourer ces structures de contrôle par des accolades, afin que les variables qui y sont déclarées ne restent visibles qu'au sein de ces structures. Mais deux éléments nous on fait juger cela inutile.

Premièrement en *Lea* les variables sont déclarées uniquement au début du programme, aucune en cours. Deuxièmement chaque nom de variable ou de label possède un identifiant unique, celui de la structure associée. Nous ne pouvons donc pas nous retrouver dans un cas où une même variable serait déclarée plusieurs fois.

Enfin, pour tester notre implémentation et détecter des erreurs potentielles, nous avons créé un fichier de test *prog-perso.lea*. Il contient une liste d'exemples plus ou moins complexes, avec plusieurs structures imbriquées, qui permettent après compilation de vérifier l'exactitude du code *C* produit.

2/ Approfondissement du sujet

Une fois l'implémentation d'une base fonctionnelle pour ces différentes structures réalisée et testée, nous nous sommes lancés dans quelques améliorations du projet, afin de rendre celui-ci plus intéressant.

2/a) La structure *switch* optimisée

Nous avons commencé par faire quelques recherches sur la façon dont un compilateur (notamment le compilateur *gcc*) réalise des optimisations de code. Nous sommes tombés sur des éléments intéressants concernant la structure *switch*, et avons décidé de nous en inspirer.

Il apparaît ainsi deux types de switch dans notre implémentation. La version choisie au moment de produire le code *C* dépend d'un critère, à savoir si les valeurs des différents *case* du switch prises dans l'ordre sont contiguës.

Notre première version, dans le cas où ces valeurs ne sont pas contiguës (par exemple dans l'ordre 4, 8, 2, 10, …), est celle déjà implémenté de base, c'est-à-dire que tous les *case* sont testés les uns après les autres (via un *if*) jusqu'à tomber sur un résultat correct ou arriver au cas *default*.

Notre seconde version, lorsque les valeurs sont contiguës (par exemple : 0, 1, 2, ..., 10, 11) permet d'optimiser le processus.

En effet, nous créons dans ce cas un tableau d'adresse, rempli avec les adresses des labels correspondant aux instructions associées à chaque *case*. La dernière case de ce tableau est remplie avec l'adresse du label correspondant aux instructions du *default*, ou avec l'adresse du label de fin du *switch* si celui-ci ne possède pas de *default*.

Ensuite, un simple test est effectué pour vérifier que l'expression du switch est supérieure au nombre de case, auquel cas on produit un *goto* avec la dernière adresse contenue du tableau, sinon on produit un *goto* avec l'adresse du tableau indexée par l'expression elle-même.

En revanche, cette solution implique d'utiliser une extension du compilateur *gcc*, à savoir l'utilisation du symbole '*&&*' devant un label pour en récupérer l'adresse. Cela à pour conséquence que notre code n'est pas portable en dehors de l'utilisation du compilateur *gcc*, mais nous avons jugé l'utilisation de ce compilateur assez prédominante pour accepter cette contrainte.

Le choix de la version du *switch* à utiliser se fait dans le fichier « *stmSWITCH* » au début de la méthode *generateCode()*, grâce à l'appel à la méthode *areCasesValuesContiguous()* implémentée par nos soins. Elle permet de détecter si oui ou non les valeurs des *case* sont contiguës ou non et renvoie un booléen en conséquence.

2/b) Ajout du type *string* pour la structure *switch*

Nous avons modifié le tableau *typeCodes* des fonctions *checkType()* dans les fichiers *« stmSWITCH »* et *« stmCASE »* pour qu'ils acceptent les expressions de type *String*.

Ainsi un test est effectué au début de *generateCode()* du fichier « *stmSWITCH()* » pour vérifier si l'expression du switch est du type *String* ou non. Dans le cas où elle le serait, nous modifions le code produit en C pour qu'il reste valide, en produisant par exemple du code utilisant la fonction *strcmp* pour tester l'égalité entre les chaînes de caractères.

Notre structure *switch* ainsi que les *case*s produites en *C* acceptent tout type de chaîne de caractère, notamment des chaînes vides, avec un ou plusieurs espaces, contenant tous les caractères standards, etc.

2/c) Structure for avec arguments vides

Dernière modification, nous avons modifié le fichier « Parser.cup » pour que la structure *for* autorise le fait d'avoir d'un premier et troisième argument vide. Par conséquent, le fichier « stmFOR » a été légèrement ajusté pour ne générer le code de ces arguments que s'ils sont présents.