PLD COMP

H4233

AL JAMOUS Myriam, BOUSLIMI Roua,CHEVALIER Mathieu, GUEDDOUDA Bachir, MAISON Benjamin, SERPINET Gaspard, SUDLOW James

Sommaire

01

Organisation

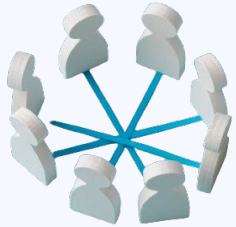
Organisation humaine

Etape 1 : Conception et planification

Etape 2 : Analyse et conception de la grammaire

Etape 3 : Développement itératif

Etape 4 : Tests et validation

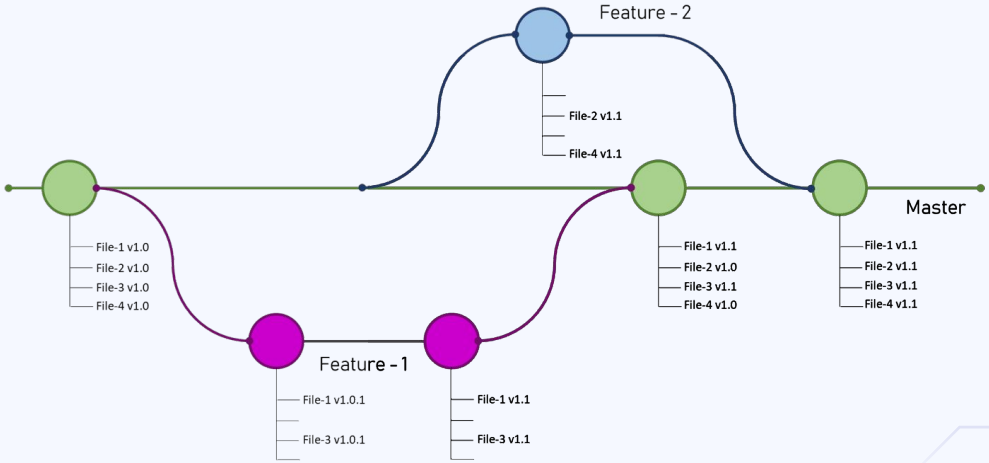


Organisation humaine

- Répartition des tâches par affinité
- Vérification de l'avancement dans le README.md
- Webhook intégration pour communiquer l'avancement
- Tour de table à 8h pour voir les choses faites/à faire/difficultés

Gestion de projet

Git -> Une branche par fonctionnalité
-> Merge de la branche à plusieurs (Pull request)



Gestion de projet

CI/CD -> vérifier compilation (du compilateur)

-> vérifier les tests (éviter regression)

-> être impartial sur le passage des tests

-> s'assurer de la qualité de main

Gestion de projet

Makefile (help, tests, verify, ...)

```
$ make help
clean               clean repo
doc                 génère la documentation avec doxygen
gui                 graphical interface
help                show this help
ifcc                compilation
tests               start all tests with python
verify              check a specific compilation
```

Changement de l'ordre et ajout de couleurs dans ifcc-test.py

Gestion de projet

```
.
├── ifcc-test.py
├── ifcc-wrapper.sh
├── Makefile
├── output.txt
└── testfiles
    ├── 10_special_tests
    ├── 11_assignment
    ├── 12_address
    ├── 14_break_continue
    ├── 1_declaration
    ├── 2_arithmetic
    ├── 3_bitwise
    ├── 4_comparision
    ├── 5_unary&lazy
    ├── 6_char
    ├── 7_function
    ├── 8_block
    └── 9_control_structures
```

Structure des fichiers

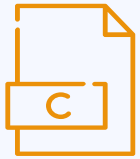
```
.
├── config
│   ├── config.mk → ubuntu.mk
│   ├── DI.mk
│   ├── fedora.mk
│   └── ubuntu.mk
├── doc
│   └── image.doxy
├── ifcc.g4
├── include
│   ├── CodeGenVisitor.h
│   ├── Error.h
│   ├── IR.h
│   ├── SymbolGenVisitor.h
│   └── Type.h
├── Makefile
├── README.md
└── src
    ├── CodeGenVisitor.cpp
    ├── CodeGenVisitorExpr.cpp
    ├── IR.cpp
    ├── main.cpp
    └── SymbolGenVisitor.cpp
```

02

Architecture

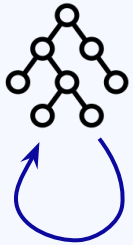
Architecture générale

Code source



1. Parsing

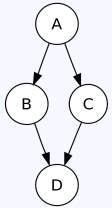
AST



2. Génération de la table des symboles

3. Génération de l'IR

IR

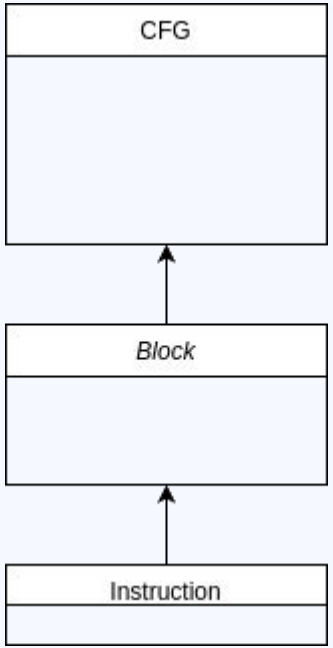


4. Génération de code

Assembleur x86

Intermediate Representation (IR)

- Séparation de la génération d'instructions et de la traduction en assembleur
- Facilite la compilation multicible
- Facilite la mise en place d'optimisations



03

Fonctionnalités

Déclaration

- Possible de faire plusieurs déclarations en une ligne avec certaines affectations
- On ne peux pas faire de re-déclaration
- Vérifie qu'une variable est déclarée (mais pas qu'elle est initialisée) avant d'être utilisé
- Vérifie qu'une variable est utilisée si elle est déclarée
- Les déclarations sont dans des scopes particuliers

Opérateurs arithmétiques

- Support des cinq opérateurs arithmétiques de base : addition (+), soustraction (-), multiplication (*), division (/), et modulo (%)
- Les règles de précédence arithmétique sont strictement appliquées
- Le compilateur peut analyser et évaluer des expressions arithmétiques complexes.

tests > testfiles > 2_arithmetic > C 26_soustraction.c > ...

```c
int main() {
    int a;
    a = 42;
    int b = 1;
    b = a - b;
    return b;
}
```

tests > testfiles > 2_arithmetic > C 36_many_op.c > ...

```c
int main() {
    int a;
    a = 4*(3+2)/3*4+3%2*(2-5);
    return a;
}
```

Opérateurs d’affectation et d’incrémentation

- **Affectation = :** déplace la valeur du membre de droite dans celle du membre de gauche avec un mov

- **+= et -= :** Se basent sur l'implémentation de la déclaration et des opérateurs +, -
- ***= et /= et %= :** Se basent sur l'implémentation de la déclaration et des opérateurs *, /, %

- **Pré-incrémentation/décrémentation :** interprète ++a en a=a+1 et on retourne a
- **Post-incrémentation/décrémentation :** on crée une variable temp égale à a, on incrémente a et on retourne la temp

Opérateurs de comparaison

- **== et != :** on compare les 2 var et on utilise “sete”, “setne” dans le registre al qui lui affecte la valeur 1 si la condition est vérifiée

- **<, <=, >, >= :** on compare les 2 var et on utilise “setl”, “setle”, “setg”, “setge” dans le registre al qui lui affecte la valeur 1 si la condition est vérifiée

Opérateurs unaires

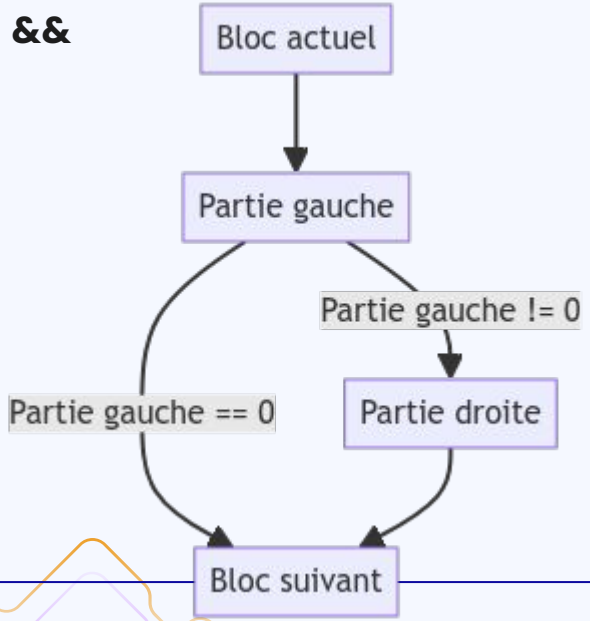
- **Opération !**
  - Comparaison avec 0
  - Stockage de l'égalité

- **Opération ~**
  - Utilisation de l'opération "not" en assembleur

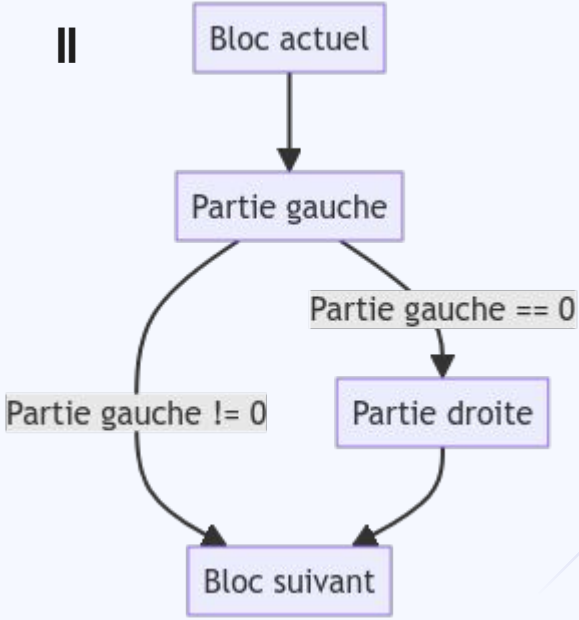
- **Opération -**
  - Utilisation de "neg" en assembleur

Opérateurs paresseux

&&



||



Opérateurs binaires et décalages

Left shift <<

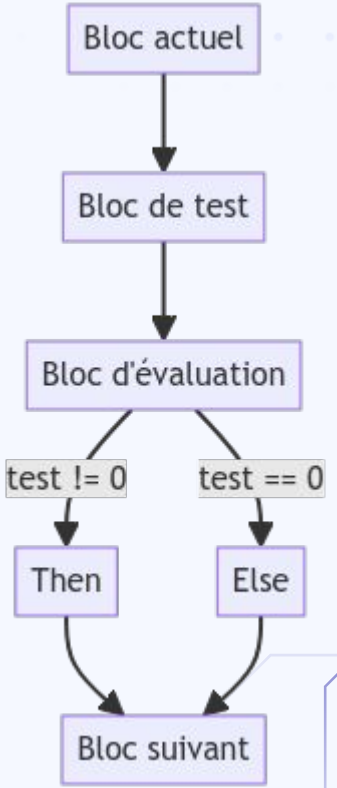
Right shift >>

And &

Or |

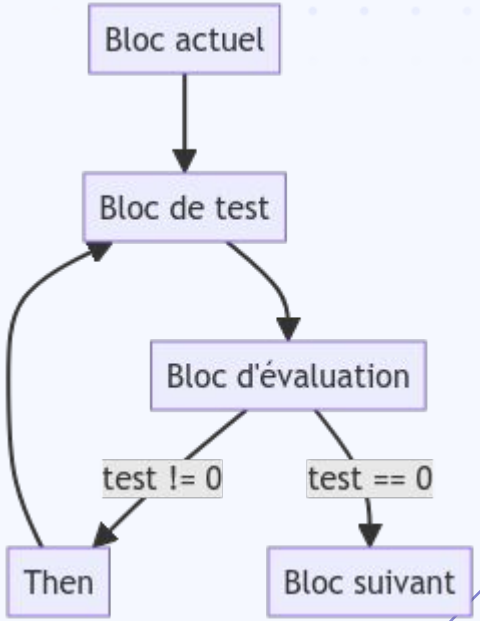
Structures de contrôle - IF

- Création de 4 ou 5 blocs
- Visite du bloc de test et d'évaluation
- Visite du then et du else
- Passage au bloc suivant qui devient notre bloc "actuel"
- Cas particulier du IF dans un while



Structures de contrôle - WHILE

- Création de 4 blocs
- Visite du bloc de test puis du bloc d’évaluation
- Visite du bloc then
- Création du bloc suivant qui devient le bloc “actuel”



Break, continue

- **Continue**
  - Jump directement au bloc de test du while
  - Implique de connaître le bloc du test dans tous les blocs “enfants”

- **Break**
  - Jump directement au bloc suivant du while
  - Implique de connaître le bloc suivant dans tous les blocs “enfants”

Structure de blocs

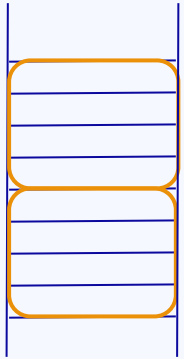
- Stockage des blocs sous forme arborescente (table de hachage qui contient les relations de parentés entre blocs)
- Assignation d'un nom unique aux variables : <nom>#<fonction>_<bloc>
- Gestion de la portée des variables et du shadowing

Type “char”

size(char)=4

+ Seulement la conversion ascii
+ Pas de conversion en assembleur (movsbl, ...)

- Pas d’optimisation mémoire



Putchar & getchar

- Des fonctions qui peuvent être appelés de l'assembleur sans implémentation
- Un encodage en ASCII
- Passage de variable dans eax

Fonctions

- Il faut modifier le rsp
- Puis mettre les variables dans les registres
- Puis fait le call
- La fonction est déclaré avant et implementé après le main
- Return avec eax

04

Démo !

Merci !