Реалізація патерну Visitor у мові C++ за допомогою std::variant

Попов Руслан (студент 1 курсу, гр. KI-23-1), Карпенко Надія

Факультет фізики, електроніки та комп'ютерних систем, Дніпровський національний університет імені Олеся Гончара, проспект Гагаріна, 72, м. Дніпро

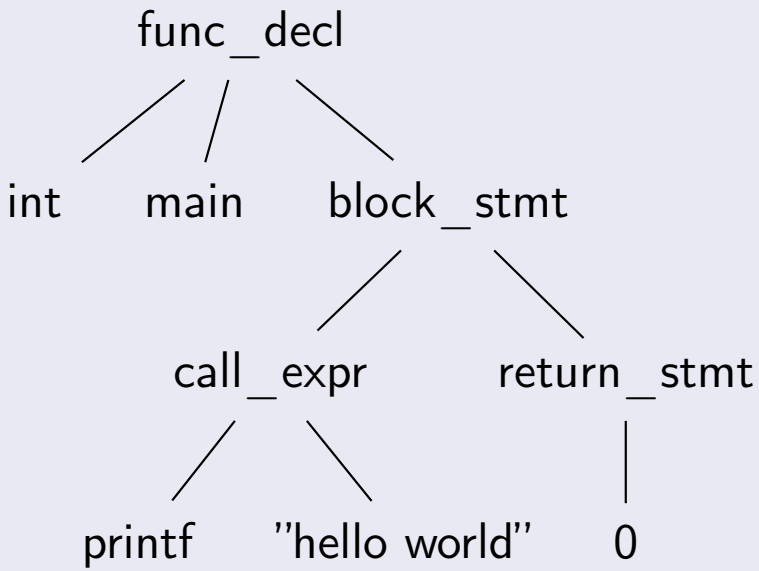
MEICS, November 2023

Дерево абстратного синтаксису

Вихідний код:

```
int main()
{
    printf("hello world");
    return 0;
}
```

Результат парсингу (спрощено):



Функції над ДАС

print: $AST \rightarrow String$

print $Var(a) =$ "a"

print $UnaryOp(-, Var(b)) =$ "$-b$"

eval: $AST \rightarrow Value$

eval $Integer(7) = 7$

eval $BinOp(Integer(2), +, Integer(2)) = 4$

optimize: $AST \rightarrow AST$

optimize $BinOp(Var(a), *, 4) = BinOp(Var(a), <<, 2)$

Перша спроба розв'язання expression problem

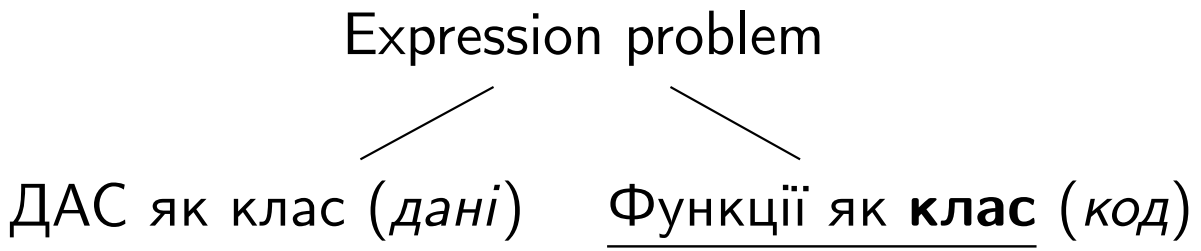
Інтерфейс Node

```
struct Node {
    virtual ~Node() {}
    virtual std::string ToString() = 0;
    virtual int Eval() = 0;
    virtual Node* Optimize() = 0;
};
```

Бінарний вираз

```
struct BinOp : Node {
    Expr* left, right;
    BinOpType op;
    ...
    virtual std::string ToString();
    virtual int Eval();
    virtual Node* Optimize();
};
```

Класичний підхід: патерн Visitor



Вузол ДАС

```
struct BinOp : Node {
    Expr* left, right;
    BinOpType op;

    virtual void Accept(Visitor* vis) {
        vis->VisitBinOp(this);
    }
};
```

Visitor

```
struct Visitor {
    virtual void VisitInteger(Integer* expr) = 0;
    virtual void VisitUnaryOp(UnaryOp* expr) = 0;
    virtual void VisitBinOp(BinOp* expr) = 0;
}
```

Проблеми з патерном Visitor

- Не легкий у розумінні.
- Треба робити багато речей для додавання нового вузла ДАС:
  1. Додати метод `Visit` до інтерфейсу `Visitor`.
  2. Створити метод `Accept` у новому вузлі ДАС.

Питання

Чи можна реалізувати цей патерн легше?

std::variant у C++

До C++17

```
// A simplified example.
struct SumType {
    int AsInteger();
    bool IsInteger();

    ValueType GetType();

private:
    ValueType type;
    union {
        int num;
        ...
    } as;
};
```

З C++17

```
#include <variant>

// Greatly reduced code size.
using SumType =
    std::variant<int, ...>;
```

std::visit у C++

Декларація std::visit починаючи з C++17

```
template <class Visitor, class... Variants>
constexpr /* ... */ visit(Visitor&& vis, Variants&&... vars);
```

Приклад використання:

```
struct MyVisitor {
    std::string operator()(int arg)   { return "integer"; }
    std::string operator()(bool arg)  { return "boolean"; }
};

int main() {
    std::variant<int, bool> var = 10;
    MyVisitor vis;
    std::cout << std::visit(vis, var) << std::endl;
    return 0;
}
```

Наш підхід до expression problem

Представити абстрактні вузли ДАС як варіанти

```
// A greately simplified example.
using Expr = std::variant<Int, Unary, Binary, ...>;
using Stmt = std::variant<If, While, Return, ...>;
```

```
struct Printer {
    std::string operator()(Int& expr);
    ...
    std::string operator()(If& stmt);
    ...
    std::string PrintExpr(Expr* expr) {
        return std::visit(*this, *expr);
    }
};
```

Деякі недоліки нашого способу

- `std::variant` доступний з C++17 $\rightarrow$ ранні стандарти не підтримуються.
- `std::variant` використовує шаблонне метапрограмування $\rightarrow$ складні повідомлення про помилки компіляції.
- Не використовуйте `Expr` всередені `Expr` або `Stmt` всередені `Stmt` $\rightarrow$ структура усередені самої себе.

Висновки

Кінець доповіді

Дякую за увагу!

Додаткова інформація

Детальний вихідний код: https://github.com/InAnYan/AstVisitor.
Електронна пошта: popov_ro@fffeks.dnu.edu.ua.

Автори

Попов Руслан (студент 1 курсу, гр. KI-23-1), Карпенко Надія.
Факультет фізики, електроніки та комп'ютерних систем, Дніпровський національний університет імені Олеся Гончара, проспект Гагаріна, 72, м. Дніпро