Experiment in Compiler Construction

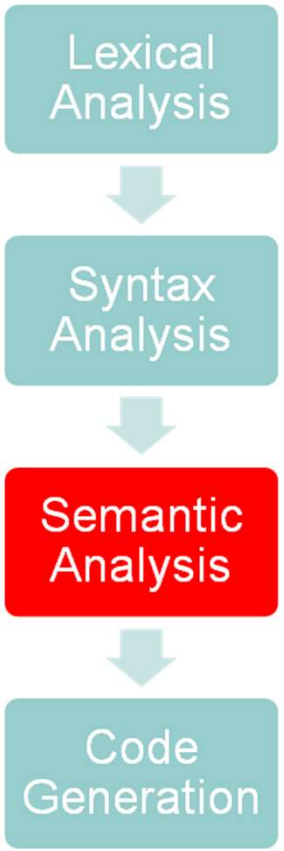
Phân tích ngữ nghĩa (1)

Viện Công nghệ Thông tin và Truyền thông
Đại học Bách khoa Hà nội

Nội dung

- Tổng quan về phân tích ngữ nghĩa
- Bảng ký hiệu
- Xây dựng bảng ký hiệu
- Một số hàm phụ trợ
- Nhiệm vụ thực hành

Phân tích ngữ nghĩa là gì?



- Phân tích cú pháp chỉ kiểm tra cấu trúc ngữ pháp hợp lệ của chương trình
- Những yêu cầu khác ngoài cấu trúc ngữ pháp:
  - "x" là tên một biến hay một hàm?
  - "x" đã được định nghĩa chưa?
  - "x" được định nghĩa ở đâu?
  - Biểu thức "a+b" có kiểu nhất quán không?
  - …
- Phân tích ngữ ngữ nghĩa trả lời các câu hỏi đó để làm rõ hơn ngữ nghĩa của chương trình.

Nhiệm vụ của một bộ phân tích ngữ nghĩa

- Quản lý thông tin về các định danh
  - Hằng
  - Biến
  - Kiểu người dùng định nghĩa
  - Chương trình con (hàm, thủ tục)
- Kiểm tra một số luật ngữ nghĩa
  - Phạm vi định danh
  - Nhất quán kiểu

Bảng ký hiệu (1)

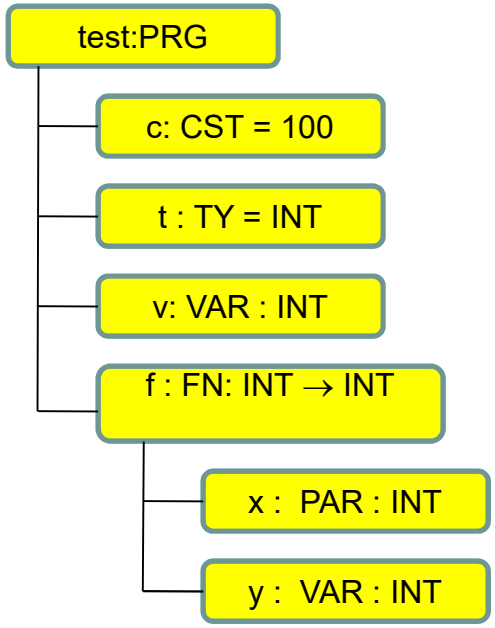
- Lưu trữ thông tin về các định danh trong chương trình và các thuộc tính của chúng
  - Hằng: {tên, kiểu, giá trị}
  - Kiểu người dùng định nghĩa: {tên, kiểu thực tế}
  - Biến: {tên, kiểu}
  - Hàm: {tên, các tham số hình thức, kiểu trả về, các khai báo địa phương}
  - Thủ tục: {tên, các tham số hình thức, các khai báo địa phương)
  - Tham số hình thức: {tên, kiểu, tham biến/tham trị}

Bảng ký hiệu (2)

- Trong chương trình dịch KPL, bảng ký hiệu được biểu diễn theo cấu trúc phân cấp

```
PROGRAM test;
CONST c = 100;
TYPE t = Integer;
VAR v : t;
FUNCTION f(x : t) : t;
  VAR y : t;
BEGIN
  y := x + 1;
  f := y;
END;

BEGIN
  v := 1;
  WriteLn (f(v));
END.
```



Xây dựng bảng ký hiệu (1)

- Các thành phần của bảng ký hiệu

```
// Bảng ký hiệu
struct SymTab_ {
  // Chương trình chính
  Object* program;
  // Phạm vi hiện tại
  Scope* currentScope;
  // Các đối tượng toàn cục như
  // hàm WRITEI, WRITEC, WRITELN
  // READI, READC
  ObjectNode *globalObjectList;
};
```

```
// Phạm vi của một block
struct Scope_ {
  // Danh sách các đối tượng trong
  // block
  ObjectNode *objList;
  // Hàm, thủ tục, chương trình
  // tương ứng block
  Object *owner;
  // Phạm vi bao ngoài
  struct Scope_ *outer;
};
```

Xây dựng bảng ký hiệu (2)

- Bảng ký hiệu ghi nhớ block hiện đang duyệt trong biến `currentScope`
- Mỗi khi dịch một hàm hay thủ tục, phải cập nhật giá trị của `currentScope`

```
void enterBlock(Scope* scope);
```

- Mỗi khi kết thúc duyệt một hàm hay thủ tục phải chuyển lại `currentScope` ra block bên ngoài

```
void exitBlock(void);
```

- Đăng ký một đối tượng vào block hiện tại

```
void declareObject(Object* obj);
```

Hằng số và Kiểu (1)

```
// Phân loại kiểu

enum TypeClass {
  TP_INT,
  TP_CHAR,
  TP_ARRAY
};

struct Type_ {
  enum TypeClass typeClass;
  // Chỉ sử dụng cho kiểu mảng
  int arraySize;
  struct Type_ *elementType;
};
```

```
// Hằng

struct ConstantValue_ {
  enum TypeClass type;
  union {
    int intValue;
    char charValue;
  };
};
```

Hằng số và Kiểu (2)

- Các hàm tạo kiểu

```
Type* makeIntType(void);
Type* makeCharType(void);
Type* makeArrayType(int arraySize, Type* elementType);
Type* duplicateType(Type* type)
```

- Các hàm tạo giá trị hằng số

```
ConstantValue* makeIntConstant(int i);
ConstantValue* makeCharConstant(char ch);
ConstantValue*
  duplicateConstantValue(ConstantValue* v);
```

Đối tượng (1)

```
// Phân loại ký hiệu

enum ObjectKind {
  OBJ_CONSTANT,
  OBJ_VARIABLE,
  OBJ_TYPE,
  OBJ_FUNCTION,
  OBJ_PROCEDURE,
  OBJ_PARAMETER,
  OBJ_PROGRAM
};
```

```
// Thuộc tính của đối tượng trên bảng ký hiệu

struct Object_ {
  char name[MAX_IDENT_LEN];
  enum ObjectKind kind;
  union {
    ConstantAttributes* constAttrs;
    VariableAttributes* varAttrs;
    TypeAttributes* typeAttrs;
    FunctionAttributes* funcAttrs;
    ProcedureAttributes* procAttrs;
    ProgramAttributes* progAttrs;
    ParameterAttributes* paramAttrs;
  };
};
```

Thuộc tính của đối tượng (1)

```
struct ConstantAttributes_ {
  ConstantValue* value;
};

struct VariableAttributes_ {
  Type *type;
  // Phạm vi của biến (sử dụng cho pha sinh mã)
  struct Scope_ *scope;
};

struct TypeAttributes_ {
  Type *actualType;
};

struct ParameterAttributes_ {
  // Tham biến hoặc tham trị
  enum ParamKind kind;
  Type* type;
  struct Object_ *function;
};
```

Thuộc tính của đối tượng (2)

```
struct ProcedureAttributes_ {
  struct ObjectNode_ *paramList;
  struct Scope_* scope;
};

struct FunctionAttributes_ {
  struct ObjectNode_ *paramList;
  Type* returnType;
  struct Scope_ *scope;
};

struct ProgramAttributes_ {
  struct Scope_ *scope;
};

// Lưu ý: các đối tượng tham số hình thức vừa được đăng ký trong danh
   sách tham số (paramList), vừa được đăng ký trong danh sách các đối
   tượng được định nghĩa trong block (scope->objList)
```

Đối tượng (2)

- Tạo một đối tượng hằng số

```
Object* createConstantObject(char *name);
```

- Tạo một đối tượng kiểu

```
Object* createTypeObject(char *name);
```

- Tạo một đối tượng biến

```
Object* createVariableObject(char *name);
```

- Tạo một đối tượng tham số hình thức

```
Object* createParameterObject(char *name
                              enum ParamKind kind;
                              Object* owner;);
```

Đối tượng (3)

- Tạo một đối tượng hàm

```
Object* createFunctionObject(char *name);
```

- Tạo một đối tượng thủ tục

```
Object* createProcedureObject(char *name);
```

- Tạo một đối tượng chương trình

```
Object* createProgramObject(char *name);
```

Giải phóng bộ nhớ

- Giải phóng kiểu

```
void freeType(Type* type);
```

- Giải phóng đối tượng

```
void freeObject(Object* obj)
```

- Giải phóng danh sách đối tượng

```
void freeObjectList(ObjectNode* objList)
void freeReferenceList(ObjectNode* objList)
```

- Giải phóng block

```
void freeScope(Scope* scope)
```

Hỗ trợ gỡ rối

- In thông tin kiểu

```
void printType(Type* type);
```

- In thông tin đối tượng

```
void printObject(Object* obj, int indent)
```

- In danh sách danh sách đối tượng

```
void freeObjectList(ObjectNode* objList, int indent)
```

- In block

```
void printScope(Scope* scope, int indent)
```

Nhiệm vụ thực hành

- Xây dựng bảng ký hiệu
- Các tệp mã nguồn
  1. `Makefile`
  2. `symtab.h, symtab.c`
  3. `debug.h, debug.c`
  4. `main.c`

Nhiệm vụ: hãy lập trình cho những hàm được đánh dấu TODO (trong tệp `symtab.c`)