# Министерство науки и высшего образования Российской Федерации

## Федеральное государственное бюджетное образовательное учреждение высшего образования

## «Новосибирский государственный технический университет»

NSTU_Logo_blue

## Кафедра теоретической и прикладной информатики

### Расчетно-графическое задание по дисциплине «Языки программирования и методы трансляции»

**Методы диагностики и исправления ошибок. Ошибки, связанные с употреблением различных типов данных**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| сигма градиент синий1 | Факультет: | ПМИ |  |  |
| Группа: | ПМ-63 |  |  |
| Студенты: | Майер В.  Шепрут И. |  |  |
| Преподаватель: | Еланцева И.Л. |  |  |

Новосибирск

2019

1. Цель работы

* Используя конспект лекций и рекомендованную научную литературу, изучить материалы по заданной теме;
* Написать реферат;
* Написать программу на исследованных методах, включив их в спроектированный транслятор.

1. Теоретическая часть

Тип данных – фундаментальное понятие теории программирования. Тип данных определяет множество значений, набор операции, которое можно применять к таким значениям и, возможно, способ реализации хранения значений и выполнения операций. Любые данные, которые оперируют программы, относятся к определенным типам.

Существуют 3 метода использования типов данных:

1. Использовать явно выраженные не статических типов. Т.е. типы всех значений в программе известны во время компиляции. Это привносит избыточность в язык и увеличивает число проверок, выполняемых в процессе компиляции. Например,   
   int i = 3;

Тут мы указываем что идентификатор i является целого типа и проверяем является ли константа 3 целого типо, если это так, то в дальнейшем мы ее запишем, иначе проверим есть ли возможность приведения типа, если возможности нет, то выдаем ошибку.

1. Использовать неявно выраженных или динамических типов. Эти типы не известны до тех пор пока не наступит время прогона программы. Например,

var i, j;

Означает, что i и j могут принимать значение любого типа. Такое присваивание, как **i = j**, всегда является допустимым.

Однако операция **i + j** может быть допустима или нет в зависимости от текущих типов i и j. Такая проверка должна включаться в объектный код, что неизбежно замедляет объектную программу.

1. Суть третьего метода заключается в том, что существует только один тип, который можно считать набором битов, а выражения **A + B** означает сложение наборов битов. Программисту могут быть известны различные типы, они неизвестны программе или реализации. Задача программиста – не допустить попыток сложения логического значения и целого числа. Так же во время компиляции и дебага невозможно проверить типы.

В С++ реализована 1 метод, также там используются ключевое слово auto, который использует инициализатор для установления типа идентификатору. Это происходит во время компиляции (при объявлении) и в дальнейшем не позволяет его менять.

Для хранения типов используется таблица символов. И диагностика ошибок происходит во время синтаксического анализа кода. И мы можем определить такие ошибки как:

* Использование необъявленной переменной
* Несогласования типов(не существует явного и неявного преобразования)

К неявным преобразованиям относятся те преобразования которые выполняет сам компилятор. Например преобразование из INT16 в INT32, в этом случае мы передадим все число, однако преобразовании из INT32 в INT16 будет происходить потеря точности, путем отсечения старших байтов. Это не будет ошибкой, но компилятор должен выдавать информацию о таком неявном привидении как предупреждение. В некоторых языках программирования такого неявное преобразование(с потерей точности) и вовсе не существует и приходится пользоваться явным.

Явное преобразование, это то преобразование которое устанавливает сам пользователь.

1. **Программная реализация**

В нашей программе мы используем явно выраженные не статические типы(метод 1), у нас нет явного преобразования типов, но программа сообщает о неявном с потерей точности(из float в int).

//Обработка тикущего состояние

switch (state)

{

case 12: //Добавление тип структуры

nameStr = tables.getStr(token);

struc = tables.add(nameStr, TABLE\_STRUCTURES);

break;

break;

case 49: //Объевление переменной

{

auto& arg = tables.get<Identifier>(token);

leftexper = token; //Выставляем как левую сторону выражения

if (type == TYPE\_STRUCT) //Если тип структура

{

auto& argStruc = tables.get<Structure>(struc); //То всем полям, которые найдем в таблице

for (auto& i : argStruc.elems) //Задаем тип из структуры

{

auto tok = tables.find(tables.getStr(token) + "." + i.name);

if (tok)

{

auto& argtok = tables.get<Identifier>(tok);

argtok.type = i.type;

}

}

arg.nameStruct = nameStr;

}

if (type != TYPE\_NONE)

{ //Задаем тип переменной

if (arg.type != TYPE\_NONE) Error("\"" + tables.getStr(token) + "\" ind was announced"); //Ошибка переменная уже была объявлена

arg.type = type;

}

else

if (tables.get<Identifier>(token).type == TYPE\_NONE) Error("\"" + tables.getStr(token) + "\" ind not announced"); //Ошибка переменная не былоа объявлена

else type = tables.get<Identifier>(token).type, nameStr = tables.get<Identifier>(token).nameStruct;

ret.push\_back(token);

}

break;

case 73: // =

{

auto& arg = tables.get<Identifier>(leftexper);

arg.isInitialized = true;

treestart = true;

oper.push(token);

}

break;

//Структуры

case 88: //id продолжение

{

ret.push\_back(token);

auto& arg = tables.get<Identifier>(token);

if (arg.type == TYPE\_STRUCT) Error("Cannot cast type \"" + arg.nameStruct + "\" in expression");

if (type == TYPE\_INT && arg.type == TYPE\_FLOAT) Warning("loss of accuracy is possible");

if (!arg.isInitialized && arg.type != TYPE\_STRUCT) Error("Not initializated varible \"" + tables.getStr(token) + "\"");

}

break;

case 89: //Константа

{

auto& arg = tables.get<Constant>(token);

string name = tables.getStr(token);

Type b;

name.find(".") == -1 ? b = TYPE\_INT : b = TYPE\_FLOAT;

if (type == TYPE\_INT && b == TYPE\_FLOAT) Warning("loss of accuracy is possible");

arg.type = b;

ret.push\_back(token);

}

break;

case 93:// Продолжение выражения

if (type == TYPE\_STRUCT) Error("Cannot cast expression to \"" + nameStr + "\"");

break;

case 78:

{

leftexper = token; // на тот случай если станет левой частью выражения

ret.push\_back(token);

auto& arg = tables.get<Identifier>(token);

if (type == TYPE\_STRUCT && arg.type == TYPE\_STRUCT && nameStr != arg.nameStruct) Error("Cannot cast type \"" + nameStr + "\" from \"" + arg.nameStruct + "\"");

if (type == TYPE\_STRUCT && arg.type != TYPE\_STRUCT) Error("Cannot cast type \"" + nameStr + "\ from " + (arg.type == TYPE\_INT ? "\" int\"" : "\" float\""));

if (type != TYPE\_STRUCT && arg.type == TYPE\_STRUCT) Error("Cannot cast type \"" + type /\*(type == TYPE\_INT ? "int\ to" : "float\ to")\*/ + arg.nameStruct + "\"");

if (type == TYPE\_INT && arg.type == TYPE\_FLOAT) Warning("loss of accuracy is possible");

if (!arg.isInitialized && arg.type != TYPE\_STRUCT) Error("Not initializated varible \"" + tables.getStr(token) + "\"");

}

break;

case 28: //Объявление тип int

type = TYPE\_INT;

break;

case 29: //Объявление тип float

type = TYPE\_FLOAT;

break;

case 30: //Объявление тип структуры

type = TYPE\_STRUCT;

nameStr = tables.getStr(token);

struc = tables.find(nameStr , TABLE\_STRUCTURES);

break;

case 23:

case 46:

case 43: //окончание объявления

//case 53:

type = TYPE\_NONE;

case 54:

if (treestart)

{

treestart = false;

while (oper.size())

{

ret.push\_back(oper.top());

oper.pop();

}

out = true;

}

else

ret.clear();

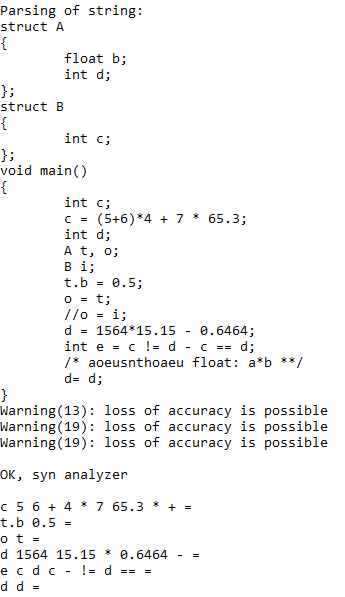
break;

default:

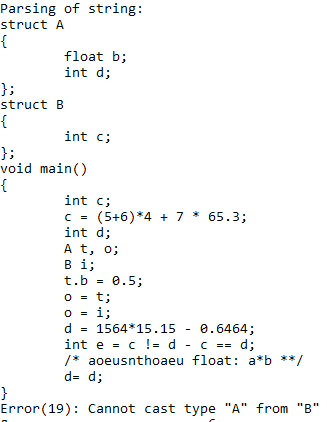
break;

}

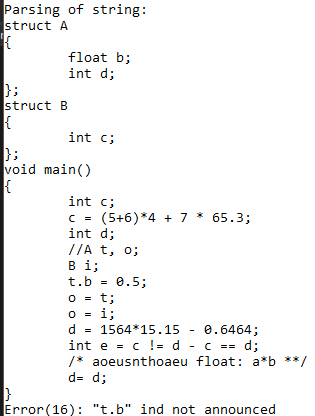
1. **Тестирование программы**
2. *Неявное приведение типа*



1. *Прирванивание разных структур*



1. *Использование необъявленной переменной*



1. **Библиографический список**

*Еланцева И.Л., Полетаева И.А.* Языки программирования и методы трансляции. Раздел «Методы трансляции». Конспект лекций. – Новосибирск: Изд-во НГТУ, 2007.

*Хантер Р.* Проектирование и конструирование компиляторов. М.: Финансы и статистика, 1984.