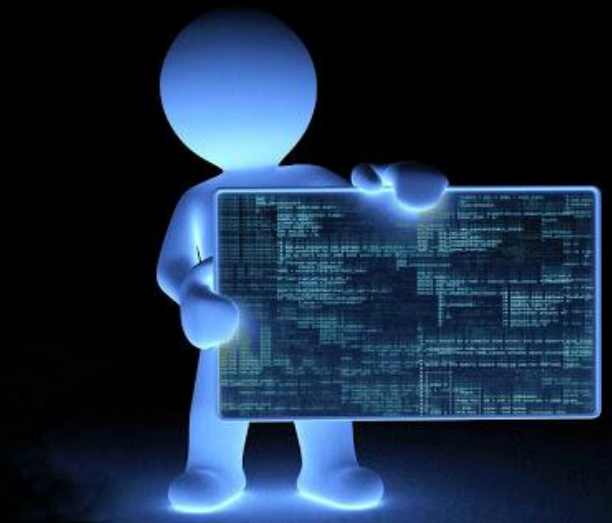
Dead code elimination

LLVM

Выполнили студенты группы ИС-741:

- Рамус Евгений
- Карасев Алексей
- Вашов Алексей
- Сысоев Максим



Удаление мёртвого кода

- В теории компиляторов **удалением мёртвого кода** называется оптимизация, удаляющая мёртвый код.

- **Мёртвым кодом** называют код, исполнение которого не влияет на вывод программы, все результаты вычисления такого кода являются мёртвыми переменными, то есть переменными, значения которых в дальнейшем в программе не используются.

Примеры мёртвого кода

- Переменные, значения которых в дальнейшем в программе не используются.
- Данная оптимизация имеет эффект только при взаимодействии с **SSA** - регистрами.

```
int foo()
{
    int a = 24;
    int b;

    b = a + 3; /* Бесполезный код */
    return a;
}
```

```
int foo()
{
    int a = 24;

    return a;
}
```

Преимущества

1) **Уменьшение:**
   - размера IR программы
   - времени работы программы

2) **Упрощение** кода для дальнейших оптимизаций

DCE & LLVM

В **LLVM** технология **DCE** реализована на основе алгоритма **Mark & Sweep.**

Алгоритм выполняется в 2 этапа:

- Фаза расставления меток
- Фаза развертки

https://llvm.org/doxygen/DCE_8cpp_source.html

Mark & Sweep
Фаза расставления меток

В **первой фазе** сборщик мусора находит и помечает все достижимые объекты.

Объект называется **достижимым**, если на него указывает поле какого-нибудь другого достижимого объекта.

Mark & Sweep
Фаза расставления меток

Объекты, к которым программа может обратиться напрямую, называются **корнями**.

Корни – это локальные переменные на стеке или глобальные переменные, указывающие на объекты в куче.

```
function MARK(q)
    if q is marked then
        return
    end if
    mark q

    for all p referenced by q do
        MARK(p)
    end for
end function
```

Mark & Sweep
Фаза развертки

Во **второй фазе** выполняется обход всех объектов в куче и освобождение тех из них, которые не помечены как достижимые.

Для корректной работы при следующих сборках алгоритм также обнуляет все пометки о достижимости.

```
function SWEEP
    for all q in heap do

        if q is not marked then
            free q
        else
            unmark q
        end if
    end for
end function
```

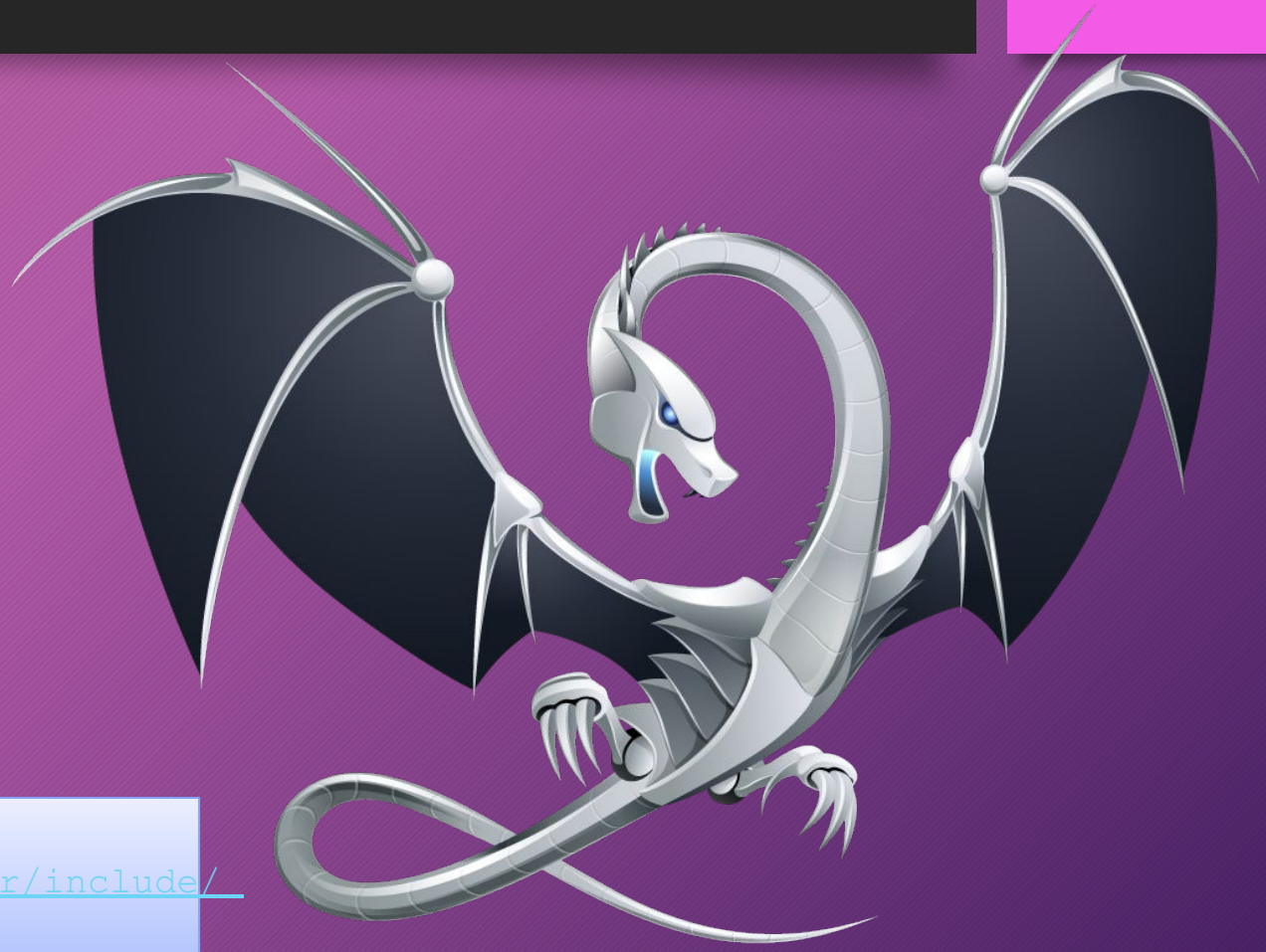
```
function DCE
    for all root r do
        MARK(r)
    end for
    SWEEP
end function
```

Устранение мертвого кода в LLVM

- Объекты, подлежащие удалению:
  - **AllocaInst** – выделение памяти в стеке;
  - **LoadInst** – чтение из памяти;
  - **GetElementPtrInst** – указатели для доступа к элементам массива и структур;
  - **SelectInst** – используется для выбора одного значения на основе условия;
  - **ExtractElementInst** – извлекает один элемент из значения векторного типа;
  - **InsertElementInst** – вставляет один элемент в векторный тип;
  - **ExtractValue** – извлекает значение поля-члена из агрегированного значения;
  - **InsertValue** – вставляет значение в поле элемента в агрегированном значении;
  - Бинарные операции
  - Операции сравнения
  - Преобразование типов

Устранение мертвого кода в LLVM

- Объекты, не подлежащие удалению:
  - **ReturnInst** – возвращает значение из функции;
  - **SwitchInst** – конструкция Switch;
  - **BranchInst** – условный и безусловный переход;
  - **CallInst** – вызов функции;
  - **StoreInst** – инструкция для хранения в памяти.



- **LLVM instructions definitions:**
  https://github.com/llvm-mirror/llvm/blob/master/include/llvm/IR/Instruction.def

DCE Example

```
$clang -S -O3 -emit-llvm -mllvm -disable-llvm-optzns dce.c
```

```c
int foo(int x, int y)
{
    int a = x + y;

    a = 1;

    return a;
}
```

```
; Function Attrs: nounwind uwtable
define dso_local i32 @foo(i32 %x, i32 %y) #0 {
entry:
  %x.addr = alloca i32, align 4
  %y.addr = alloca i32, align 4
  %a = alloca i32, align 4
  store i32 %x, i32* %x.addr, align 4, !tbaa !2
  store i32 %y, i32* %y.addr, align 4, !tbaa !2
  %0 = bitcast i32* %a to i8*
  call void @llvm.lifetime.start.p0i8(i64 4, i8* %0) #2
  %1 = load i32, i32* %x.addr, align 4, !tbaa !2
  %2 = load i32, i32* %y.addr, align 4, !tbaa !2
  %add = add nsw i32 %1, %2
  store i32 %add, i32* %a, align 4, !tbaa !2
  store i32 1, i32* %a, align 4, !tbaa !2
  %3 = load i32, i32* %a, align 4, !tbaa !2
  %4 = bitcast i32* %a to i8*
  call void @llvm.lifetime.end.p0i8(i64 4, i8* %4) #2
  ret i32 %3
}
```

```
$opt dce.ll -S -dce -print-after-all
```

```
; Function Attrs: nounwind uwtable
define dso_local i32 @foo(i32 %x, i32 %y) #0 {
entry:
  %x.addr = alloca i32, align 4
  %y.addr = alloca i32, align 4
  %a = alloca i32, align 4
  store i32 %x, i32* %x.addr, align 4, !tbaa !2
  store i32 %y, i32* %y.addr, align 4, !tbaa !2
  %0 = bitcast i32* %a to i8*
  call void @llvm.lifetime.start.p0i8(i64 4, i8* %0) #2
  %1 = load i32, i32* %x.addr, align 4, !tbaa !2
  %2 = load i32, i32* %y.addr, align 4, !tbaa !2
  %add = add nsw i32 %1, %2
  store i32 %add, i32* %a, align 4, !tbaa !2
  store i32 1, i32* %a, align 4, !tbaa !2
  %3 = load i32, i32* %a, align 4, !tbaa !2
  %4 = bitcast i32* %a to i8*
  call void @llvm.lifetime.end.p0i8(i64 4, i8* %4) #2
  ret i32 %3
}
```

```
$opt dce.ll -S -sroa -dce -print-after-all
```

```
*** IR Dump After SROA ***
; Function Attrs: nounwind uwtable
define dso_local i32 @foo(i32 %x, i32 %y) #0 {
entry:
  %add = add nsw i32 %x, %y
  ret i32 1
}
*** IR Dump After Dead Code Elimination ***
; Function Attrs: nounwind uwtable
define dso_local i32 @foo(i32 %x, i32 %y) #0 {
entry:
  ret i32 1
}
```