TECHNISCHE UNIVERSITÄT DARMSTADT

FOLLOW-UP

Embedded Systems & Applications

Virtuelle Destruktoren in C++

TECHNISCHE UNIVERSITÄT DARMSTADT

- Virtuelle Destruktoren werden nicht erzwungen!
- Aber wenn man sie vergisst, passieren Dinge.

```
class BaseClass {
      ~BaseClass() {
            /* Do nothing here */
      }
};
class DerivedClass: public BaseClass {
      DerivedClass() :
                  member(0) {
            member = new int[10];
      }
      ~DerivedClass() {
            delete[] member;
      }
      int* member;
};
BaseClass* obj = new DerivedClass();
delete obj; // <- MEMORY LEAK
            // "~BaseClass" is called, but not "~DerivedClass" -> member will not be deleted!
```

ABBILDUNG VON SPRACHKONSTRUKTEN

Unions

TECHNISCHE UNIVERSITÄT DARMSTADT

```c
#include <stdio.h>

typedef union {
      long long ll;
      long      l;
      int       i;
      short     s;
      char      c;
} ints_u;

int main() {
      ints_u U;
      U.ll = 0xDEADBEEFCAFEBABELL;

      printf ("U.s = %d\n", U.s);
      return 0;
}
```

- Typ des größten Elements
- Frontend erzeugt bitcasts für Zugriff auf andere Elemente

```
; ModuleID = 'union.ll'
target datalayout = "e-p:64:64:64-i1:8:8-i8:8:8-i16:16:16-
i32:32:32-i64:64:64-f32:32:32-f64:64:64-v64:64:64-v128:128:128-
a0:0:64-s0:64:64-f80:128:128-n8:16:32:64-S128"
target triple = "x86_64-apple-macosx10.9.0"

%union.ints_u = type { i64 }

@.str = private unnamed_addr constant [10 x i8] c"U.s = %d\0A
\00", align 1

; Function Attrs: nounwind ssp uwtable
define i32 @main() #0 {
  %U = alloca %union.ints_u, align 8
  %1 = bitcast %union.ints_u* %U to i64*
  store i64 -2401053089206453570, i64* %1, align 8
  %2 = bitcast %union.ints_u* %U to i16*
  %3 = load i16* %2, align 2
  %4 = sext i16 %3 to i32
  %5 = call i32 (i8*, ...)* @printf(i8*
getelementptr inbounds ([10 x i8]* @.str, i32 0, i32 0), i32 %4)
  ret i32 0
}
```

λ-Funktionen (aka Closures)

TECHNISCHE UNIVERSITÄT DARMSTADT

Neues Sprachfeature in C++11

```
$ clang++ --std=c++11 lambda.cpp
```

```
#include <iostream>

using namespace std;

int main() {
      int x;
      cin >> x;

      auto func = [&] (int i) -> void { cout << "Hello, No. " << i << ", you typed " << x << endl; };
      func(42);

      auto func2 = [] (const string &s) -> void { cout << "Have a " << s << " day!" << endl; };
      func2("nice");

      return 0;
}
```


Embedded Systems & Applications

λ-Funktionen (aka Closures)

TECHNISCHE UNIVERSITÄT DARMSTADT

Neues Sprachfeature in C++11

```
$ clang++ --std=c++11 lambda.cpp
```

```
#include <iostream>

using namespace std;

int main() {
      int x;
      cin >> x;

      auto func = [&] (int i) -> void { cout << "Hello, No. " << i << ", you typed " << x << endl; };
      func(42);

      auto func2 = [] (const string &s) -> void { cout << "Have a " << s << " day!" << endl; };
      func2("nice");

      return 0;
}
```

Capture specification

[]	„Nichts“
[&]	„By-Reference“
[=]	„By-Value“

Embedded Systems & Applications

λ-Funktionen (aka Closures)

TECHNISCHE UNIVERSITÄT DARMSTADT

Neues Sprachfeature in C++11

```
$ clang++ --std=c++11 lambda.cpp
```

```
#include <iostream>

using namespace std;

int main() {
      int x;
      cin >> x;

      auto func = [&] (int i) -> void { cout << "Hello, No. " << i << ", you typed " << x << endl; };
      func(42);

      auto func2 = [] (const string &s) -> void { cout << "Have a " << s << " day!" << endl; };
      func2("nice");

      return 0;
}
```

Parameterliste

Capture specification

[]	„Nichts“
[&]	„By-Reference“
[=]	„By-Value“

Embedded Systems & Applications

λ-Funktionen (aka Closures)

TECHNISCHE UNIVERSITÄT DARMSTADT

Neues Sprachfeature in C++11

```
$ clang++ --std=c++11 lambda.cpp
```

```
#include <iostream>

using namespace std;

int main() {
      int x;
      cin >> x;

      auto func = [&] (int i) -> void { cout << "Hello, No. " << i << ", you typed " << x << endl; };
      func(42);

      auto func2 = [] (const string &s) -> void { cout << "Have a " << s << " day!" << endl; };
      func2("nice");

      return 0;
}
```

Parameterliste

Rückgabewert (optional)

Capture specification

[]	„Nichts“
[&]	„By-Reference“
[=]	„By-Value“

Embedded Systems & Applications

λ-Funktionen (aka Closures)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Lowering passiert im Frontend

- Für (fast) jede λ-Funktion wird eine neue Klasse erzeugt:
  - Der ()-Operator wird entsprechend (Parameter, Rückgabewert, Rumpf) überschrieben.
  - Ein Konstruktor zur Übergabe der gecapture'ten Variablen wird erzeugt.
  - Ausnahme: λs, die keine Variablen capturen, werden zu Funktionen.

Quelle (und gute Erklärung): http://www.cprogramming.com/c++11/c++11-lambda-closures.html

λ-Funktionen (aka Closures)

TECHNISCHE UNIVERSITÄT DARMSTADT

Darstellung in LLVM (Ausschnitt aus ~ 1000 Zeilen LLVM Assembler…)

```
auto func = [&] (int i) -> void { cout << "Hello, No. " << i << ", you typed " << x << endl; };
func(42);
```

```
define i32 @main() #0 {
  %x = alloca i32, align 4
  %func = alloca %class.anon, align 8            ; %class.anon = type { i32* }
  %func2 = alloca %class.anon.0, align 1         ; %class.anon.0 = type { i8 }
  …
  %3 = getelementptr inbounds %class.anon* %func, i32 0, i32 0
  store i32* %x, i32** %3, align 8
  call void @"_ZZ4mainENK3$_0clEi"(%class.anon* %func, i32 42)
```

λ-Funktionen (aka Closures)

TECHNISCHE UNIVERSITÄT DARMSTADT

Darstellung in LLVM (Ausschnitt aus ~ 1000 Zeilen LLVM Assembler…)

```
auto func = [&] (int i) -> void { cout << "Hello, No. " << i << ", you typed " << x << endl; };
func(42);
```

Closure-Objekt

```
define i32 @main() #0 {
  %x = alloca i32, align 4
  %func = alloca %class.anon, align 8          ; %class.anon = type { i32* }
  %func2 = alloca %class.anon.0, align 1       ; %class.anon.0 = type { i8 }
  …
  %3 = getelementptr inbounds %class.anon* %func, i32 0, i32 0
  store i32* %x, i32** %3, align 8
  call void @"_ZZ4mainENK3$_0clEi"(%class.anon* %func, i32 42)
```


Embedded Systems & Applications

λ-Funktionen (aka Closures)

TECHNISCHE UNIVERSITÄT DARMSTADT

Darstellung in LLVM (Ausschnitt aus ~ 1000 Zeilen LLVM Assembler…)

```
auto func = [&] (int i) -> void { cout << "Hello, No. " << i << ", you typed " << x << endl; };
func(42);
```

Closure-Objekt

```
define i32 @main() #0 {
  %x = alloca i32, align 4
  %func = alloca %class.anon, align 8          ; %class.anon = type { i32* }
  %func2 = alloca %class.anon.0, align 1       ; %class.anon.0 = type { i8 }
  …
  %3 = getelementptr inbounds %class.anon* %func, i32 0, i32 0
  store i32* %x, i32** %3, align 8
  call void @"_ZZ4mainENK3$_0clEi"(%class.anon* %func, i32 42)
```

Adresse von x capturen
(statt Konstruktor-Aufruf)

Embedded Systems & Applications

λ-Funktionen (aka Closures)

TECHNISCHE UNIVERSITÄT DARMSTADT

Darstellung in LLVM (Ausschnitt aus ~ 1000 Zeilen LLVM Assembler…)

```
auto func = [&] (int i) -> void { cout << "Hello, No. " << i << ", you typed " << x << endl; };
func(42);
```

Closure-Objekt

```
define i32 @main() #0 {
  %x = alloca i32, align 4
  %func = alloca %class.anon, align 8          ; %class.anon = type { i32* }
  %func2 = alloca %class.anon.0, align 1       ; %class.anon.0 = type { i8 }
  …
  %3 = getelementptr inbounds %class.anon* %func, i32 0, i32 0
  store i32* %x, i32** %3, align 8
  call void @"_ZZ4mainENK3$_0clEi"(%class.anon* %func, i32 42)
```

Adresse von x capturen
(statt Konstruktor-Aufruf)

Die Methode aufrufen

Embedded Systems & Applications

λ-Funktionen (aka Closures)

TECHNISCHE UNIVERSITÄT DARMSTADT

```
auto func2 = [] (const string &s) -> void { cout << "Have a " << s << " day!" << endl; };
func2("nice");
```

```
invoke void
  @„_ZZ4mainENK3$_1clERKNSt3__112basic_stringIcNS0_11char_traitsIcEENS0_9allocatorIcEEEE"(
  %class.anon.0* %func2, %"class.std::__1::basic_string"* %1)
  to label %10 unwind label %11
```

Davor: String-Initialisierung
Danach: Exception Handling

Embedded Systems & Applications

PROGRAMMIEREN MIT LLVM

„Navigation“ in der IR

TECHNISCHE UNIVERSITÄT DARMSTADT

- Argumente
  - `Instruction *I = … ; I->getOperand(1);`
  - `BranchInst *Br = …; Br->getCondition();`
  - `CallInst *Ci = …; Ci->getArgOperand(4); Ci->getCalledFunction();`

- CFG
  - `Function *F = …; F->getEntryBlock();`
  - `BasicBlock *BB = …;`
    - `for (pred_iterator PI = pred_begin(BB), PE = pred_end(BB); PI != PE; PI++) { …`
    - `for (succ_iterator SI = succ_begin(BB), SE = succ_end(BB); SI != SE; SI++) { …`
    - `TerminatorInst *TI = BB->getTerminator();`
    - `if (BranchInst *BI = dyn_cast<BranchInst>(TI))`
      `BI->getSuccessor(1) // „false“`

Embedded Systems & Applications

„Navigation“ in der IR

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Module *M = …;
for (Module::iterator MI = M->begin(), ME = M->end();
     MI != ME; MI++) {
    Function *F = MI;
    for (Function::iterator FI = F->begin(), FE = F->end();
         FI != FE; FI++) {
        BasicBlock *BB = FI;
        for (BasicBlock::iterator BBI = BB->begin(), BBE = BB->end();
             BBI != BBE; BBI++) {
            Instruction *I = BBI;

            assert (I->getParent() == BB);
            assert (I->getParent()->getParent() == F);
            assert (I->getParent()->getParent()->getParent() == M);
        }
    }
}
```


Embedded Systems & Applications

„Navigation“ in der IR

TECHNISCHE UNIVERSITÄT DARMSTADT

- Jeder `Value` hat eine Liste von `Usern`

```
Value *V = …;
for (use_iterator UI* = V->use_begin(), UE = V->use_end();
     UI != UE; UI++) {
  User *U = UI;
```

- `User` sind alle IR-Elemente, die andere Werte referenzieren können
  - Hauptsächlich Instructions und Konstanten

- Beispiel: Alle (direkten) Aufrufstellen einer Funktion F

```
for (use_iterator UI* = F->use_begin(), UE = F->use_end();
     UI != UE; UI++) {
  if (CallInst *CI = dyn_cast<Function>(UI)) {
    …
```


Embedded Systems & Applications

InstVisitor

TECHNISCHE UNIVERSITÄT DARMSTADT

```
class MyVisitor : public InstVisitor<MyVisitor> {
public:
    void visitAdd(BinaryOperator &I) { … }

    void visitStoreInst(StoreInst &I) { … }
    void visitTerminatorInst(TerminatorInst &I) { … }
};

// somewhere
Function &F = …;
MyVisitor V; V.visit(F);
```

- Überschreibbare Methoden für Opcodes, spezifische Instruktionen, und Klassen von Instruktionen.
- Makromagie, keine `accept(…)`-Methode in Instruction-Hierarchie.

Embedded Systems & Applications

(Post)DominatorTree

TECHNISCHE UNIVERSITÄT DARMSTADT

- Im Pass die Analyse anfordern

```
void getAnalysisUsage(AnalysisUsage &AU) const {
    AU.addRequired<DominatorTree>();
}
```

- Verwendung

```
DominatorTree &DT = getAnalysis<DominatorTree>();
if (DT.dominates(BB1, BB2)) …

DomTreeNode *DTN1 = DT[BB3], *DTN2;
DTN2 = DTN1->getIDom();

DT.findNearestCommonDominator(DTN1, DTN2);
```


Embedded Systems & Applications

LoopInfo

- Liefert natürliche Schleifen (= ein Eintrittspunkt)
- Im Pass die Analyse anfordern

```
AU.addRequired<LoopInfo>();
```

- Sehr sinnvoll: Voher `-loop-simplify` laufen lassen
- Verwendung

```
LoopInfo &LI = getAnalysis<LoopInfo>();
for (LoopInfo::iterator …
  // Toplevel Loops

Loop *L = LI.getLoopFor(BB);
unsigned d = LI.getLoopDepth(BB);
```

- Verwendung der Loop-Objekte
  - Zugriff auf ausgezeichnete Blöcke (falls eindeutig)