Advanced Models and Programs:

MicroC, pointer programming, type system, compilation

Peter Sestoft
Mon 2009-02-16*

Agenda

• Concepts: Pointers, lvalue, rvalue
• MicroC parsing challenges
• MicroC abstract syntax
• MicroC compilation schemes
The essence of C: Pointers

• Main innovations of C (1972) over Algol 60:
  – Structs, as in COBOL and Pascal
  – Pointers, pointer arithmetics, pointer types, array indexing as pointer indexing
  – Using { } for blocks, as in C++, Java, C#
• Java and safe C# lack pointers and are completely different from C/C++
• Except for the core language, which is Algol with { } instead of begin end.

Pointer basics

• A pointer p refers to a storage location
• The dereference expression *p
  – may mean the content of the location as in *p+4
  – may mean the storage location itself, as in *p = x+4
• The pointer that points to x is &x
• Pointer arithmetics: *(p+1) is the storage location after *p
• If p equals &a[0] then *(p+i) equals p[i] equals a[i], so arrays are pointers
Using pointers for return values

- Example test12.c, computing square(x):

```c
void main() {
    int res;
    square(5, &res);
    write res;
    square(res, &res);
    square(res, &res);
    write res;
}

void square(int x, int *p) {
    *p = x*x;
}
```

**MicroC abstract syntax**

![Diagram of MicroC abstract syntax]
Concepts

- Expression statements
- Storage model
- Activation records (frames)
  - Base pointer bp
  - Return address
  - Array layout (B rather than C)
- Lvalue and Rvalue

Expression statements in C, C++, Java and C#

- The “assignment statement”
  \[ x = 2+y; \]
  is really an expression
  \[ x = 2+y \]
  followed by a semicolon
  ;
- The semicolon means: ignore value

```c
public class ExprStatement : Statement {
    private readonly Expression e;
    public override void Compile(CEnv env, Generator gen) {
        e.Compile(env, gen);
        gen.Emit(new INCSP(-1));
    }
}
```
Storage model

- The store is an indexable stack
  - Bottom: global variables at fixed addresses
  - Plus, a stack of activation records

- An activation record is an executing function
  - return address
  - old base pointer
  - local variables
  - temporaries

Array layout

- An array int arr[5] consists of
  - its 5 int elements
  - a pointer to arr[0]

- This is the array representation of B, the predecessor to C
Lvalue and rvalue of an expression

- Rvalue is “normal” value, right-hand side of assignment: 17, true
- Lvalue is “location”, left-hand side of assignment: x, a[2]
- In e1=e2, expression e1 must have lvalue
- Where else must an expression have lvalue in C#? In C?

<table>
<thead>
<tr>
<th></th>
<th>lvalue</th>
<th>rvalue</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>a[2]</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>*p</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>x+2</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>&amp;x</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

MicroC parsing

- Variable declaration syntax
- Lvalue and rvalue expressions
- Right-associative operators
C variable type declarations

<table>
<thead>
<tr>
<th>Declaration</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>int n</td>
<td>n is an integer</td>
</tr>
<tr>
<td>int *p</td>
<td>p is a pointer to integer</td>
</tr>
<tr>
<td>int ia[3]</td>
<td>ia is array of 3 integers</td>
</tr>
<tr>
<td>int *ipa[4]</td>
<td>ipa is array of 4 pointers to integers</td>
</tr>
<tr>
<td>int (*iap)[3]</td>
<td>iap is pointer to array of 3 integers</td>
</tr>
<tr>
<td>int *(*ipap)[4]</td>
<td>ipap is pointer to array of 4 pointers to ints</td>
</tr>
</tbody>
</table>

The Unix program cdecl helps understand C types:

cdecl> explain int *(*ipap)[4]
declare ipap as pointer to array 4 of pointer to int
cdecl> declare n as array 7 of pointer to pointer to int
text**n[7]

Parsing C variable declarations

- Cumbersome because declarations are mixfix, eg: int *x[4]
- Parser trick: Use C# delegate type T2V that transforms Type to VarDec1:
  public delegate VarDecl T2V(Type t);
- Return a T2V and apply it to the type:
  VarDec<out VarDecl varDecl> (Type ty; T2V t2v; )
  = Typ<out t2v> VarDesc<out t2v> (varDecl = t2v(ty); )
  .
  VarDesc<out T2V t2v> (String name; )
  = (Ident<out name>
    (t2v = delegate(Type ty) { return new VarDecl(name, ty); }
    )
  | /* VarDesc<out t2v>
    (t2v = delegate(Type ty) { return t2v(new PointerType(ty)); }
    | ...
  ...
Parsing Lvalue and Rvalue expressions

- LR parsing can distinguish expressions having lvalues from those that don’t
- LL parsing cannot, and must parse assignment as `Expr "=" Expr`
- But `x+2=78;` should be illegal.
- Solution: Make grammar too general
  
  `Expr<out lhs> "=" Expr<out rhs>`
  
  and then check that lhs has lvalue:

  ```java
  e = new Assignment(lhs.MakeLvalue(), rhs);
  ```

Only accesses (x, a[2], *p) have lvalue

```java
public abstract class Expression {
    public virtual LvalueExpression MakeLvalue() {
        throw new Exception("Illegal lvalue expression");
    }
}

class LvalueExpression : Expression {
    private readonly Access access;
    public override LvalueExpression MakeLvalue() {
        return this;
    }
}

class Access : Expression {
    public override LvalueExpression MakeLvalue() {
        return new LvalueExpression(this);
    }
}

class VariableAccess : Access {
    public readonly String name;
}
```
Parsing assignment \( x = y = e \), which is right associative

- Expr. \( x = y = z = v \) should parse as \( x = (y = (z = v)) \)
- Parse as Expr AssignExpr where AssignExpr is \([ ‘=’ Expr ]\)

```java
Expr<out Expression e> (. Expression rhs; .)
= LogOrExp<out e>
[ ‘=’ Expr<out rhs>
   (. e = new Assignment(e.MakeLvalue(),
     rhs);
   .)
] .
```

Throws if no lvalue

---

MicroC checking

- Check that variables and functions are declared when used
- Check that types match
  - in arithmetic and logical expressions
  - in pointer and array indexing
  - in pointer dereferencing and address-of
  - in function parameter passing
  - in assignment expressions
Ten-minute exercise

• Assume \( p \) is pointer to \texttt{int}

• What kinds of pointer arithmetics are allowed by MicroC.cs?
  – Addition of integer: \( p+2 \)
  – Multiplication by integer: \( p*2 \)
  – Subtraction of integer: \( p-2 \)

• What should be changed to make pointer difference \( p-q \) have type \texttt{int}?

Compiling arithmetic expressions and assignment

• Compile \( e_1 + e_2 \):
  \(<e_1>\) as \texttt{rvalue}
  \(<e_2>\) as \texttt{rvalue}
  ADD

• Compile \( e_1 = e_2 \):
  \(<e_1>\) as \texttt{lvalue}
  \(<e_2>\) as \texttt{rvalue}
  STI

• \(<e_1>\) means: the result of compiling \( e_1 \)
### Compiling lvalues and rvalues

- **Compile `e` as rvalue:**
  - `<e> as lvalue
  - `LDI`

- **Compile `&e` as rvalue:**
  - `<e> as lvalue

- **Compile `x` as lvalue:**
  - `GETBP`
  - `CSTI <xoffset>`
  - `ADD`

- **Compile `e1[e2]` as lvalue:**
  - `<e1> as rvalue
  - `<e2> as rvalue
  - `ADD`

- **Compile `*e` as lvalue:**
  - `<e> as rvalue

### Compiling blocks and declarations

- **To compile a block** `{ s1 s2 ... sn }`:
  - Enter new scope
  - `<s1> <s2> ... <sn>`
  - Leave scope

- **To compile declaration** `int x:`
  - Generate code to push 0; this takes a stack place

- **To compile declaration** `int a[5]:`
  - Generate code to allocate 5 stack locations, that is, increment SP by 5
  - Generate code to compute address of the first of those locations
Statement compilation schemes

- Compile
  if (e) s1 else s2:
  
  <e> as rvalue
  IFZERO L1
  <s1>
  GOTO L2
  L1: <s2>
  L2:

- Compile
  while (e) s:
  
  L1: <e> as rvalue
  IFZERO L2
  <s>
  GOTO L1
  L2:

Ten-minute exercise

- What code should be generated for a do-while block:
  
  do s while (e) ;

- What code should be generated for a for statement:
  
  for (e1; e2; e3) s
The Micro-C stack machine

- Compiling and running Micro-C

Coco -namespace MicroC MicroC.ATG
csc MicroC.cs Scanner.cs Parser.cs

csc Machine.cs

unzip test-c.zip
MicroC test5.c check
MicroC test5.c compile

Machine a.out
Machine /trace a.out

Build
Build compiler

machine
Build stack

Compile
example

Run it

with trace

The stack of frames

- Example test5.c, computing factorial:

```c
void main() {
    int res;
    fac(3, &res);
    write res;
}

void fac(int n, int *res) {
    if (n == 0)
        *res = 1;
    else {
        int tmp;
        fac(n-1, &tmp);
        *res = n * tmp;
    }
}
```

- n is input parameter
- res is output parameter, a pointer to where to put the result
- tmp holds the result of the recursive call to fac
- &tmp gets the pointer to tmp
The code for test5.c

```
0 CALL 0 L1
3 STOP
4 L1:
4 INCP 1
6 CSTI 3
8 GETBP
9 CSTI 0
11 ADD
12 CALL 2 L2
15 INCP -1
17 GETBP
18 CSTI 0
20 ADD
21 LDI
22 PRINTI
23 INCP -1
25 INCP -1
27 RET -1
29 L2:
30 CSTI 0
32 ADD
33 LDI
34 CSTI 0
36 EQ
37 IFZERO L3
39 GETBP
40 CSTI 1
42 ADD
43 LDI
44 CSTI 1
46 STI
47 INCP -1
49 GOTO L4
51 L3:
53 GETBP
54 CSTI 0
56 ADD
57 LDI
58 CSTI 1
60 GETBP
61 LDI
62 CSTI 2
64 ADD
65 CALL 2 L2
68 INCP -1
70 GETBP
71 CSTI 1
73 ADD
74 LDI
75 GETBP
76 CSTI 0
78 ADD
79 LDI
80 GETBP
81 CSTI 2
83 ADD
84 LDI
85 MUL
86 STI
87 INCP -1
89 INCP -1
91 RET 1
93 RET 1
```

The stack of frames

- Example test5.c: computing fac(0)
- Stack frame for fac(0):

```
[ ](O: CALL 0 4)
[ 3 -999 ](4: CST 0)
[ 3 -999 0 ](8: GETBP)
[ 3 -999 0 2 ](11: ADD)
[ 3 -999 0 2 0 ](12: CALL 2 29)
[ 3 -999 0 15 2 0 2 ](29: GETBP)
[ 3 -999 0 15 2 0 2 5 ](30: CST 0)
[ 3 -999 0 15 2 0 2 5 0 ](32: ADD)
[ 3 -999 0 15 2 0 2 5 33: LDI]
[ 3 -999 0 15 2 0 2 0 ](34: CST 0)
[ 3 -999 0 15 2 0 2 0 0 ](36: EQ)
[ 3 -999 0 15 2 0 2 1 ](37: IFZERO 51)
[ 3 -999 0 15 2 0 2 ](39: GETBP)
[ 3 -999 0 15 2 0 2 5 ](40: CST 1)
[ 3 -999 0 15 2 0 2 5 1 ](42: ADD)
[ 3 -999 0 15 2 0 2 6 ](43: LDI)
[ 3 -999 0 15 2 0 2 ](44: CST 1)
[ 3 -999 0 15 2 0 2 2 1 ](46: STI)
[ 3 -999 1 15 2 0 2 5 ](47: INCP -1)
[ 3 -999 1 15 2 0 2 ](49: GOTO 91)
[ 3 -999 1 15 2 0 2 ](81: INCP 0)
[ 3 -999 1 15 2 0 2 ](83: RET 1)
[ 3 -999 1 1 ](15: INCP -1)
[ 3 -999 1 1 ](17: GETBP)
[ 3 -999 1 1 ](18: CST 0)
[ 3 -999 1 0 ](20: ADD)
[ 3 -999 1 2 ](21: LDI)
[ 3 -999 1 1 ](22: PRINTI)
[ 3 -999 1 1 ](23: INCP -1)
[ 3 -999 1 ](25: INCP -1)
[ 3 -999 ](27: RET -1)
[-999 ](3: STOP)
```

- What stack frame?

```
15 2 0 3
```

```
ret-addr  old  n  tmp
```
Computing fac(3) with test5.c

- On the whiteboard...

Exercises Wednesday/Week 4

- Write Micro-C programs
  - to sum elements of an array
  - to make histogram of the elements of array
- Change Micro-C parser and compiler to
  - Allow (==) and (!=) to compare pointers and ints
  - Add a C/C++/Java/C#-style for-loop
  - Add prefix increment (++) operator
  - Add conditional operator (?:)
- Investigate the (Java or .NET) bytecode of selection sort