

Notes from Saturday's review session kindly donated by Ramy.

topics:

syntax directed translation
garbage collection
register allocation
exceptions
small lang
object oriented- dispatch tables, object layouts

gui design

```
Form (main) {  
X MyCheck;  
<My Text> (25);  
[My Button];  
Form (SubMain) {  
    [Another Button]  
}  
}
```

Grammar without actions

Checkbox --> label
TextField --> width, initial text

ListOfStmt --> epsilon | Stmt ListOfStmt
Stmt --> Checkbox | TextField | Button | Form
Checkbox --> 'X' <text> ';' ;

TextField --> '<' <text> '>' ';' |
 '<' <text> '>' '(' <num> ')' ';' ;

Button --> '[' <text> ']' ';' ;
Form -> <Form> '(' <text> ')' '{ ListOfStmt '}'

Sample HTML Label: <input type='button'
 label='My button'>

Grammar with actions:

Checkbox --> label
TextField --> width, initial text

ListOfStmt --> epsilon [] push(" "); []
 | Stmt ListOfStmt [] String lst = peek(-1) + "\n" + peek(0);
 push(lst);
 []

Stmt --> Checkbox [] push(peek(0)); []
 | TextField [] push(peek(0)); []
 | Button [] push(peek(0)); []
 | Form [] push(peek(0)); []

```

Checkbox --> 'X' <text> ';' [] String st = "<input type='checkbox' label=' ";
                st = st + peek(-2);
                st = st + "' '>";
                push(st);
                []

```

```

TextField --> '<' <text> '>' ';' |
                '<' <text> '>' '(' <num> ')' ';' [] String st = "<input type='text' value=' ";
                st += peek(-5) + "' width=' + peek(-2) + "' '> ";
                push(st);
                []

```

```

Button --> '[' <text> ']' ';' [] String st = "<input type='button' label=' ";
                st = st + peek(-2);
                st = st + "' '>";
                push(st);
                []

```

```

Form -> <Form> '(' <text> ')' '{ ListOfStmt }' [] String st = "<Form name = ' " + peek(-4) +
                "' '> " + peek(-1) + "</ Form>";
                push(st);
                []

```

Sample parsing:

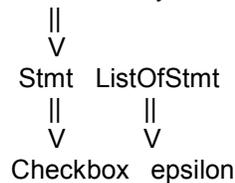
```

Form (main) {
    X mycheck;
    Form (submain) {}
}

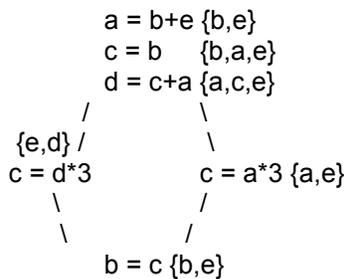
```

parse tree:

form-> tokens, name, tokens, ListOfStmt, }



Register Allocation with 3 Registers:



stack from bottom to top: b,c,a,d,e

-given that we can push all nodes onto the stack, we know there is a 3-coloring

Garbage Collection:

mark and sweep-main advantage is that it allows conservatism vis-a-vis ambiguity of ints as pointers and vice versa
stop and copy-due to this ambiguity, this scheme won't work for C
-reference counting

Given the following, which schemes work:

```
int hashVal(Object obj) {  
    1- if i call hashVal on the same object multiple times, i get the same result  
    2- for two objects that are live at the same time, they have different hash values  
    -return the address of object in memory
```

mark-n-sweep and reference counting work, but stop and copy doesn't work because the physical address of the object changes, and hence the hashVal.

Exceptions:

1) long jump- as we enter a try-catch block we push a handler's context onto the stack, so when an exception is thrown we sift through the stack to find a befitting handler. the advantage here is that we don't pay up front for exception handling, rather every time we encounter a new handler we must update the data structure, which slows execution, and when we actually throw an exception we need $O(n)$ time to find the right handler.

2) tables- we generate a table for each method, and in the table we specify lines between which a particular handler is used. This requires us to pay for exception handling with huge chunks of memory devoted to the exception tables, although with virtual memory, that's not a big problem as long as we don't use them. You don't pay for exceptions unless you throw one.

-create a new entry for each different type of exception that's handled
-JRE will select first compatible handler