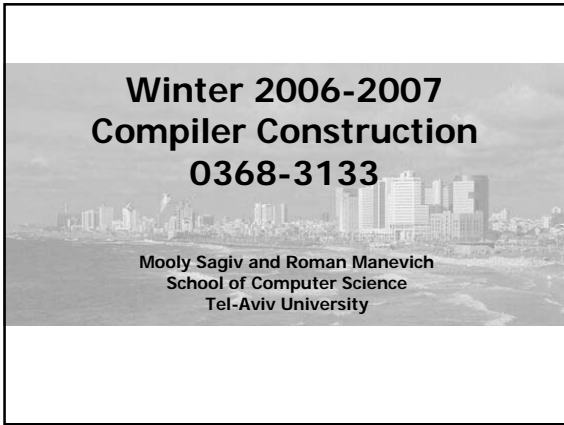


**Winter 2006-2007
Compiler Construction
0368-3133**



Mooly Sagiv and Roman Manevich
School of Computer Science
Tel-Aviv University

Who

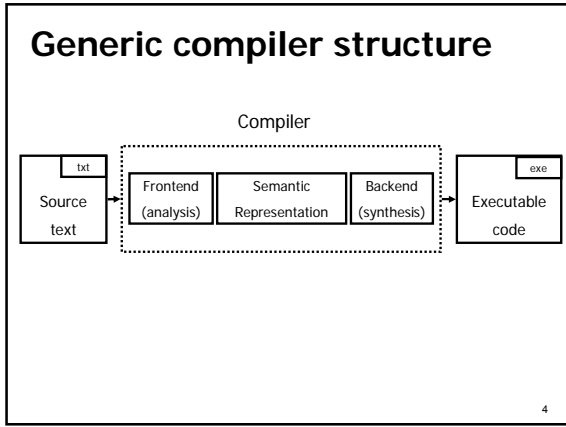
Roman Manevich
Schreiber Open-space (basement)
Tel: 640-5358
rumster@post.tau.ac.il
Wednesday 16:00-17:00
<http://www.cs.tau.ac.il/~rumster/wcc06/>

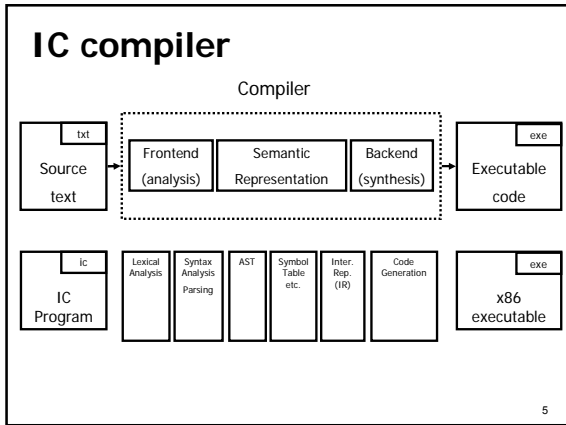
2

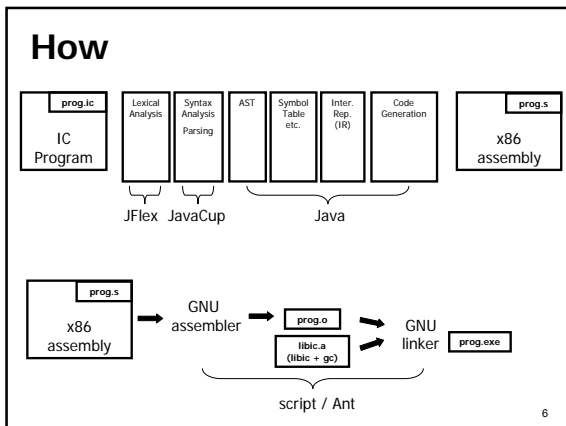
Mailing list and forum

- Mailing list web interface:
<http://listserv.tau.ac.il/archives/cs0368-3133-02.html>
<http://listserv.tau.ac.il/archives/cs0368-3133-03.html>
<http://listserv.tau.ac.il/archives/cs0368-3133-04.html>
- Activate e-mail by going to:
<https://www.tau.ac.il/newuser/>
- Forwarding, changing email address:
<https://www.tau.ac.il/forward/>
<http://www.ims.tau.ac.il/tal/>
- Not registered? Mail to listserv@listserv.tau.ac.il
with the line: subscribe CS0368-3133-0X Real Name
where X is your recitation course number (2/3/4)
- Forum:
<http://www.cs.tau.ac.il/system/forums/viewforum.php?f=46>

3

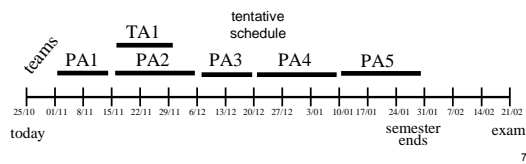






Grading and schedule

- 45% exam
- 5% theoretical assignment
- 50% project
 - 5 assignments – different weights
 - code checked both automatically and manually

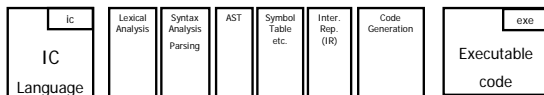


Project guidelines

- Teams of 2 or 3 students
 - Email me before next recitation
 - Subject: Compilation - new project team
 - List of members (first name, last name, id, username on nova)
 - Team-account user name
- In case of doubt – ask questions in the forum
- There is adequate time to complete assignments
- Start early and please follow directions

8

Today



- Goals:
 - IC language overview
 - See [language specification](#)
 - Understand the scope of the project

9

IC language - main features

- Strongly-typed
 - Primitive types for int, boolean, string
 - Reference types
- Object oriented
 - Objects, virtual method calls
 - Inheritance
- Memory management
 - Dynamic heap allocation of objects and arrays
 - Automatic deallocation (garbage collection)
 - Runtime safety checks
 - Null dereference
 - Division by 0
 - Array access out of bounds
- Many things are left out
 - Short implementation time
- Adapted with permission from Prof. Radu Rugina (Cornell University)

10

Unsupported features

- Access control
 - Everything is public
- Interfaces
- Downcasting
- Method overloading
(but still allow overriding)
- Exceptions
- Packages
- Multiple source files

11

IC program structure

- Program is sequence of class definitions
- Class is sequence of fields and methods
 - Static methods and *virtual* methods
 - Exactly one main method
`static void main(string[] args) {...}`
- Variables can be declared anywhere in a method
 - Check initialization before use
- Strings are primitive types
- Arrays `T[]`, `T[][]`

12

IC types

- Every class is a type
- Primitive types:
 - `int` : 1, -1, 2, -2, ...
 - `boolean` : `true`, `false`
 - `string` : "hello"
- References : `null`
- Arrays : `int [] x = new int[5]; x.length==5;`
- All variables must be declared
 - compiler infers types for expressions
- Type-safety
 - Well-typed programs do not result in runtime type errors

13

Subtyping

- Inheritance induces subtyping relation
 - Type hierarchy gives acyclic graph (forest)
 - Subtyping rules:

$$\frac{A \text{ extends } B \{ \dots \}}{A \leq B} \quad \frac{}{A \leq A} \quad \frac{A \leq B \quad B \leq C}{A \leq C} \quad \frac{}{\text{null} \leq A}$$

- Subtyping does not extend to array types
 - A subtype of B then A[] is not a subtype of B[]

14

Expressions

- Expression language
 - Every expression has a type and a value
 - Loops: `while (expr) { stmt }`
 - Conditionals: `if (expr) stmt else stmt`
 - Arithmetic operators: `+` `-` `*` `/` `%`
 - Relational comparison: `<` `>` `==` `<=` `>=`
 - Logical operators: `&&` `||`
 - Unary operators: `!` `-`
 - Assignment: `x = expr`

15

Objects

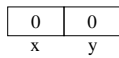
- Instances of classes are objects

```
class Point {  
    int x; // initialized to 0  
    int y;  
}
```

- new Point ()** allocates object of class **Point** on heap and initializes fields

- No arguments

- An object can be thought of as a struct (record) with a slot for each field



16

Methods

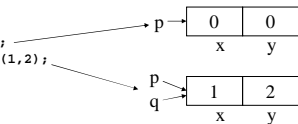
```
class Point {  
    int x;  
    int y;  
    Point movePoint(int newX, int newY) {  
        x = newX;  
        y = newY;  
        return this;  
    } -- close method  
} -- close class
```

- A class can also define methods to manipulate fields
- Methods can refer to the current object using **this**

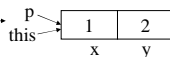
17

Example

```
class TestPoint {  
    Point p;  
    Point q;  
    boolean test() {  
        p = new Point();  
        q = p.movePoint(1,2);  
        return p==q;  
    }  
}
```



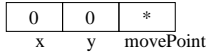
```
class Point {  
    int x;  
    int y;  
    Point movePoint(int newX, int newY) {  
        x = newX;  
        y = newY;  
        return this;  
    }  
}
```



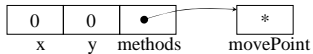
18

Method implementation

- Each object knows how to access method code
- As if object contains slot pointing to the code



- In reality implementations save space by sharing these pointers among instances of the same class

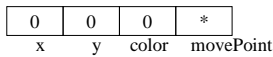


19

Inheritance example

- We can extend points to colored points:

```
class ColoredPoint extends Point {
    int color;
    Point movePoint(int newx, int newy) {
        color = 0;
        x = newx;
        y = newy;
        return this;
    }
}
```



20

Method invocation and inheritance

- Methods are invoked by dispatch
- Understanding dispatch in the presence of inheritance is a subtle aspect of OO languages

```
Point p;
p = new ColoredPoint();
p.movePoint(1,2);
```

- `p` has static type `Point`
- `p` has dynamic type `ColoredPoint`
- `p.movePoint` invokes `ColoredPoint` implementation

21

Method invocation

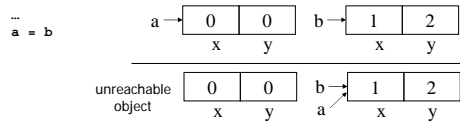
- Example: invoke method `p.movePoint(a+b, 20/2)`

1. Evaluate `p`
2. Find class of `p`
3. Find code of `movePoint`
4. Evaluate arguments `a+b, 20/2`
5. Bind `p` to `this`
6. Bind actual arguments to formal arguments
7. Run method code

22

IC memory management

- Memory allocated every time `new` is used
- Memory deallocated automatically by reclaiming unreachable objects
 - Done by a garbage collector (GC)
 - Use off-the-shelf GC



23

Library functions

- `libc` provides:
 - I/O operations
 - datatype conversions
 - system level-operations

```
class library {
    static void println(string s); // prints string s followed by a newline.
    static void print(string s); // prints string s.
    static void printi(int i); // prints integer i.
    static void printb(boolean b); // prints boolean b.
    static int readi(); // reads one character from the input.
    static string readln(); // reads one line from the input.
    static boolean eof(); // checks end-of-file on standard input.
    static int stoi(string s, int n); // returns the integer that s represents
    // or n of s is not an integer.
    static string itos(int i); // returns a string representation of i.
    static int[] stoa(string s); // an array with the ascii codes of chars in s.
    static string atos(int[] a); // builds a string from the ascii codes in a.
    static int random(int i); // returns a random number between 0 and n-1.
    static int time(); // number of milliseconds since program start.
    static void exit(int i); // terminates the program with exit code n.
}
```

24

For next week

- Split into teams
 - Send me email with team members and representative account
- Read IC language specification
 - <http://www.cs.tau.ac.il/~rumster/wcc06/icspec.pdf>
- Get acquainted with Java
 - [J2SE 1.5](#)
 - [Eclipse IDE](#)
 - More helpful links on web-site

25

See you next week

26
