

Compiler Construction 0368-3133

Rina Zviel-Girshin and Ohad Shacham
School of Computer Science
Tel-Aviv University

Staff

- Instructor: Rina Zviel-Girshin
- Grader: Paz Grimberg
- Technical Assistant: Ohad Shacham
 - Schreiber Open-space (basement room 2)
 - Wednesday 10:00 – 11:00
 - Phone #5358

2

Administrative

- <http://www.cs.tau.ac.il/research/ohad.shacham/wcc09/wcc09.html>
- Email: ohad.shacham@cs.tau.ac.il
- Forum: <https://forums.cs.tau.ac.il/viewforum.php?f=70>

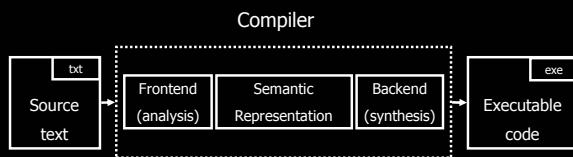
3

Administrative

- There will be no recitation on November 4th
- Alternative recitation on one of the following:
 - November 20
 - November 27
 - December 4

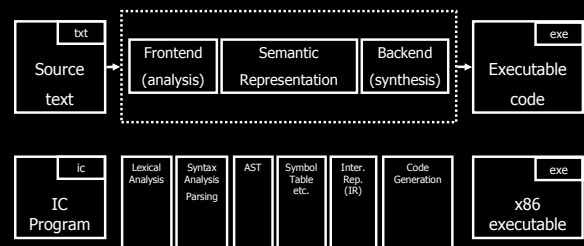
4

Generic compiler structure

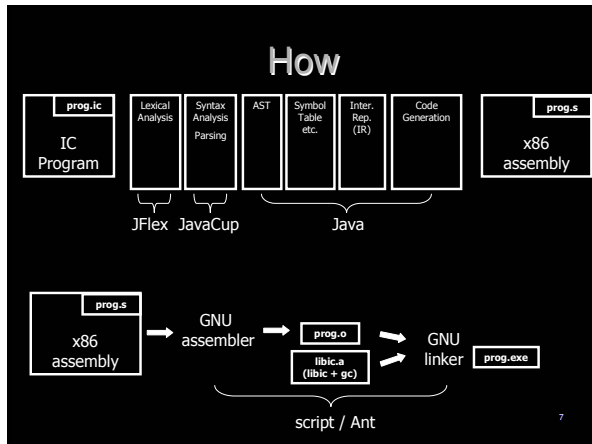


5

IC compiler Compiler

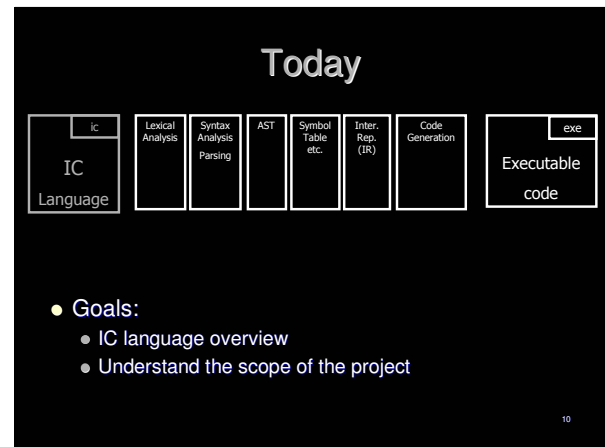


6



- ## 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
 - List of members (first name, last name, id, username on nova)
 - Team-account user name
 - There is adequate time to complete assignments
 - Start early and please follow directions
 - Submission in your home directories



- ## 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
 - Adapted with permission from Prof. Radu Rugina (Cornell University)

- ## Unsupported features
- Access control
 - Everything is public
 - Interfaces
 - Method overloading (but still allow overriding)
 - Exceptions
 - Packages

IC program structure

- Program is sequence of class definitions
- Class is sequence of fields and methods
 - Static methods
 - *virtual* methods
 - Exactly one main method
`static void main(string[] args) {...}`

13

IC program structure

- Variables can be declared anywhere in a method
 - Check initialization before use
- Object fields and Array elements are initialized
- strings are primitive types
- Arrays `T[], T[][]`

14

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

15

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[]`

16

Expressions

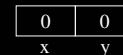
- Expression language
 - Every expression has a type and a value
 - Loops: `while (expr) { stmt }`
 - Conditionals: `if (expr) stmt else stmt`
 - Arithmetic operators: + - * / %
 - Relational comparison: < > == <= >=
 - Logical operators: && ||
 - Unary operators: ! -
 - Assignment: `x = expr`
 - break,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

17

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



18

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**

19

Example

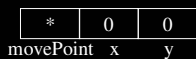
```
class TestPoint {
  Point p;
  Point q;
  boolean test() {
    p = new Point();
    q = p.movePoint(1,2);
    return p==q;
  }
}

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

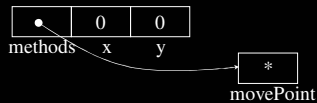
20

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



21

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;
  }
}
```

22

Method invocation and inheritance

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

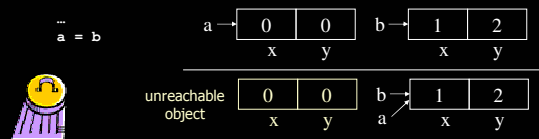
```
Point p;
p = new ColoredPoint();
p.movePoint(1,2);
```

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

23

GC 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



24

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 if s is not an integer.
    // returns a string representation of i.
    static string itos(int i); // an array with the ascii codes of chars in s.
    static int[] stoa(string s); // builds a string from the ascii codes in a.
    static string stoa(int[] a); // returns a random number between 0 and n-1.
    static int randon(int n); // number of milliseconds since program start.
    static int time(); // terminates the program with exit code n.
    static void exit(int n);
}
```

25

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/research/ohad.shacham/wcc09/doc/cspsc.pdf>
- Get acquainted with Java
 - **J2SE 1.5** (or higher)
 - **Eclipse IDE**
- Install and play with JFlex and Java Cup

26

Next week

- Lexical analysis
- JFlex
 - Lexical analyzer generator in Java
- Explain PA1

27