Announcement

Traditional competition "Semicolon" takes place on 9th May in the afternoon.

Generic data-types

how to define them

- We know that there exist generic data-types (e.g., `List`). But how to define them?
- Remark: These data-types are light-version of templates in C++, those can be indexed by anything (e.g., by number).
- They can be used if we want to create several instances with different underlying data-type.
- It is also a replacement of preprocessor-macros in C.

Generics II

- When using generics, we proceeded as with normal data-type, just we added a parameter into angle-brackets [chevrons].
- When defining generic data-type we do almost the same, i.e., we just name the parameter and behave with that as with a data-type.
- There does not have to be only one parameter, more parameters are separated by commas.
- `public class gen_cl <T> {public T variable;}`

Generic class example

```
public class my_list <T>
{    public T data;
     public my_list<T> next;
}
...
     my_list<int> x=new my_list<int>();
     x.data=new my_list<int>();
     //This would not work:
     // x.next=new my_list<double>();
```

Generic methods

not only classes may be generic

- In the function-body the parameter behaves as when defining generic class.
- This time we put chevrons containing the data-types between function-name and parameters.
- `void gen_met<T,U>(T a, U b){ Console.WriteLine("Parameters are {0},{1}.",Convert.ToString(a),Convert.ToString(b));}`

Generic method example

```
static void swapit<T>(ref T a, ref T b)
{    T tmp=a;
     a=b;
     b=tmp;
}
static void Main()
{    int a=1,b=2;
     swapit<int>(ref a, ref b);
     Console.WriteLine("a is {0}, b is {1}",a,b);
}
```

Restricting data-types

in newer versions of .NET Framework

- We may restrict the parameter using keyword `where`.
- When defining the class, we put `where` behind the chevrons,
- when defining the methods, it stands after the header.
- `class gen<T> where T:IComparable{...}`
  `//T implements function CompareTo`
- `void gener<T>(out T a) where T:new(){a=new T();}`

Operator overloading

is the same as function overloading

- We know that a function is defined by its name and argument-structure.
- Several functions with the same name may exist.
- Also there are, e.g., many types of numbers: integers, longint, double, rational numbers...
- ... and we want to add, subtract, multiply or divide them...
- without calling obscure functions like `add_two_rationals`.

Complex numbers

and operations on them, a. k. a. operator overloading

- We want to create a class representing complex numbers,...
- whose elements can be added like `c=a+b;`
- thus we overload operator $+$.
- When overloading an operator, it looks like function overloading just the name of the function is fixed (e.g., "`operator +`") `and number of arguments, too (follows from grammar of C#).`
- We may overload an operator in a class identical with at least one of its parameters (we cannot overload an operator in completely different class).
- And the functions must be `static`!

Example

Gaussian integers

```
class compl
{    public int re, im;
     public compl(int re, int im)
     {    this.re=re; this.im=im;'}
     public static compl operator +(compl a,compl b)
     {    return new compl(a.re+b.re,a.im+b.im);}
     public static compl operator *(compl a,compl b)
     {    return new compl(a.re*b.im-a.im*b.im,
               a.re*b.im+a.im*b.re);
     }
}
```

Example – continued

To make the class `compl` demonstrable, we override her a method `ToString`, too:

```
public override string ToString()
{    return ""+re+"+ "+im+"i";}
```

And let's go:

```
kompl a=new compl(1,0), b=new compl(0,1),c;
c=a+b;
Console.WriteLine(c);
Console.WriteLine(a*b);
```

Overloadable operators

We may overload the operators:
unary `!`, `~`, `++`, `--`
binary `+`, `-`, `*`, `/`, `%`, `&`, `|`, `^`, `<<`, `>>`
WE CANNOT overload mainly &&, ||, [], (*type*)*x*, +=, −=...

Games Programming

- Combinatorial game is a game of two players. State of the game is given by position of particular items. All items relevant to the game are visible for both players. I.e., combinatorial games are games with full information.
- Examples: Nim, Strange game, Draughts, Chess, Halma, Nine Men's Morris, Poisoned chocolate, Devilish darts,...
- Combinatorial games are *NOT*: Poker, Mau mau, Black jack, formula-race, Doom,...
- We focus on the playing algorithms (not on input/output).
- We expect the players to behave rationally (i.e., they want to win).

Shannon's theorem

Theorem (Shannon)

Each combinatorial game (with finite number of possible moves) has a winning strategy for at least one of the players.

Důkaz.

Sketch: Either at least one of the players may enfoce the game to start cycling (to never finish). Then he never loses and theorem holds. Or the game is finite and we examine predicates:
There exists our (1st player's) move s.t. for all moves of 2nd player there exists our move s.t.,... we win.
For all our moves there exists a move of the 2nd player s.t. for all our moves... we do not win.
Predicates are negation one to another and they are finite (finite quantification), thus they are decidable and exactly one of them

Game-graph

- For a game (its instance) we assign an oriented graph:
- Vertices represent states of the game,
- edges represent possibility of transition between states.
- Example for Nim with 1 or 2 matches (on white-board).
- Each state (vertex) may be colored according to who wins (when starting here).

Examples of graphs

- The game is represented by an oriented graph and we are moving a coin over this graph starting in a given vertex.
- We should reach one of terminal vertices; who cannot move, loses (who reaches that state, wins).
- Graph of the game is given, questionable is how to win.
- Note that each game can be represented as Devilish darts.
- Strange game: vertices are individual squares of the board.
- It is enough to say whether the player moving from current vertex wins or loses (or if there is a cycle that both players appreciate).

AND-OR-tree

- Considering a game-graph, we may create game-tree instead by splitting one state into possibly more states.
- For this tree we may ask whether there is a branch where we win.
- Such a "branch" is a subtree, s.t., after an odd step for all possibilities (even step) there is an odd step,..., i.e.,...
- either we win in the first son, or in the second, or in the third,...
- while the other player loses in the first (grand-)son and in the second son and in the third,...
- AND- and OR-gates are regularly interlacing, thus AND-OR-tree.

Games with evaluation

Definition

Game with an evaluation is a game where the result of a game is a value. One player tries to maximize this number while the other is trying to minimize it.

Definition

Game with a zero sum is a game where win for one player is a loss for the other player (and these values are the same).

That's all for today...

...thank you for your attention.