

Revision: arrays

- By creating an array, we ask the Java system to allocate a big piece of memory to hold a particular type of data.
- We need to know the number of elements to create, the size of the array, only at run-time.
- An expression like `new int[10]` creates an array.
- The expression returns a reference value. We store it in a variable of a type like `int[]`, an array reference to integer.
- We can then access an array by something like `arr[i]`, where `arr` is an array reference.

0911B L15-1

Some clarifications

- A number is associated with each the array element, from 0 to `size-1`. It is called the **index** of the element.
- An array reference is just like a simple variable. So it can be assigned if their type match. For example, if we have `int[] a, b`; then we can say `a=b` so `a` points to the same array as `b` does.
- The equality operator (`==`) does the natural thing on array references: if `a` and `b` point to the same place (or are both `null`), `a==b` returns `true`, otherwise it returns `false`.
- When an array reference is passed as method argument, the array reference is passed by value. This means that the formal argument points to the same array as the actual argument, so **methods can change values of array elements** (unlike, say, an `int`).

0911B L15-2

The logic of the game: main()

What we need to do is basically setup the game board, and then do the following until the game ends: let the current side to make a move, and switch to the other side.

At the end we print out the board again and show the winner. So the logic is as follows:

```
Setup board, current_side = WHITE
while (!endGame()) {
    allow current_side to make a move
    current_side = the other side
}
Print the board and the winner.
```

0911B L15-3

Allowing a move: allowMove()

At this time, even if we write `endGame()`, we don't know how to test it. So it is better to just let it always return `true`.

So now we have to deal with allowing input. What we need is basically printing the board, getting an input, and then try to make the move. If that fails, we print out error, and repeat.

```
// Should have something more here...
Repeat forever {
    Print out the board
    Print out a message, read Y
    Print out a message, read X
    if making move at (Y - 1, X - 1) is successful
    then break
    Print out error message and the board again
}
```

0911B L15-4

Making a move: makeMove(y,x)

What we need is to check whether the move is really in the board and whether there is nothing there before. If it's not the case, the move is invalid. Otherwise, try to flip the pieces on each direction. If none succeed it is invalid. Otherwise it is valid, and we place a piece.

```
if (offBoard(y,x) || board[y][x]!=UNFILLED)
    return false
succeed = false
succeed |= tryFlip(y,x,-1,-1)
...
succeed |= tryFlip(y,x,1,1)
if (!succeed)
    return false
board[y][x] = current_side
return true
```

0911B L15-5

Trying to flip: tryFlip(y,x,dy,dx)

To try flipping from (y, x) towards a direction (dy, dx) , we first check flipping is possible. If no flipping is possible, we return false. Otherwise we do the flipping, by stepping through $(y + dy, x + dx)$, $(y + 2dy, x + 2dx)$ and so on until we reach a piece of the same side.

```
if (!checkFlip(y,x,dy,dx,current_side))
    return false
y += dy
x += dx
while (board[y][x] is not a piece of current_side) {
    board[y][x] = current_side
    y += dy
    x += dx
}
return true
```

0911B L15-6

Checking possibility of flip: `checkFlip(y,x,dy,dx,side)`

Checking flips is very similar to actually flipping them: we step through $(y + dy, x + dx)$, $(y + 2y, x + 2x)$ and so on until we either go off the board, or we hit an empty position, or we hit a position with a piece of the one to move. We keep track of whether we have seen pieces of the other side.

```
found_other = false
Repeat forever {
  y+=dy
  x+=dx
  if (offBoard(y,x) || board[y][x]==UNFILLED)
    return false
  if (board[y][x]==side)
    break
  found_other = true
}
return found_other
```

0911B L15-7

Reminders on assignment

The soft deadline of the assignment is on the next Monday.

You should have everything needed to do the programming assignment now.

Hint on programming problem:

To convert a character (e.g. 'C') to a number (e.g. 2), you can use subtraction. E.g., 'C'-'A' gives you (int)2.

To do the reverse, add a number to 'A' and cast the result to a char. E.g. (char>('A'+2) gives you 'C'.

Debugging a program that behaves randomly might be difficult. Try to use a fixed, small array to test your program first, before subject it to real tests.

0911B L15-8