CHAPEL BASICS

Chapel Team, edited by Michelle Strout

April 3, 2025

HANDS ON: HOW TO DO THE HANDS ON

01-hello.chpl

Example codes for Chapel tutorial slides

- https://github.com/UofA-CSc-372-Spring-2025/CSc372Spring2025-CourseMaterials/tree/main/Sandboxes/ChapelTutorialExamples

Using a container on your laptop

- First, install docker for your machine and start it up (see the README.md for more info)
- Then, use the chapel-gasnet docker container

```
docker pull docker.io/chapel/chapel-gasnet    # takes about 5 minutes
cd CSc372Spring2025-CourseMaterials/Sandboxes/ChapelTutorial/
docker run --rm -it -v "$PWD":/workspace chapel/chapel-gasnet
root@589405d07f6a:/opt/chapel# cd /workspace
root@xxxxxxxxx:/myapp# chpl 01-hello.chpl
root@xxxxxxxxx:/myapp# ./01-hello -nl 1
```

PLAN

- **Announcements**
  - Final project assignments coming out ASAP
  - SA7 will be posted Friday April 4th, a couple of days late
- **Last time**
  - TopHat questions about ChapelCon tutorial and last class, aka ICA10 prep
  - Kmer counting example: file IO, maps, strings
  - Parallel processing of files
  - Overview of GPU programming support
- **Today**
  - TopHat Questions
  - ICA10 Quiz
  - Chapel Programming Basics in the context of an Nbody simulation, part I

ICA10: QUIZ ON CHAPEL SO FAR

- Read the instructions on the quiz

OUTLINE: OVERVIEW OF PROGRAMMING IN CHAPEL

- Running Example: n-body computation (Hands On)
- Variables, Constants, and Operators
- Records and Classes
- Tuples
- Arrays
- Writing out Tuples, Records, and Arrays (Hands On)

RUNNING EXAMPLE: N-BODY COMPUTATION (HANDS ON)

N-BODY IN CHAPEL (WHERE N == 5)

- A serial computation

- From the Computer Language Benchmarks Game
  - Chapel implementation in release under examples/benchmarks/shootout/nbody.chpl

- Computes the influence of 5 bodies on one another
  - The Sun, Jupiter, Saturn, Uranus, Neptune

- Executes for a user-specifiable number of timesteps

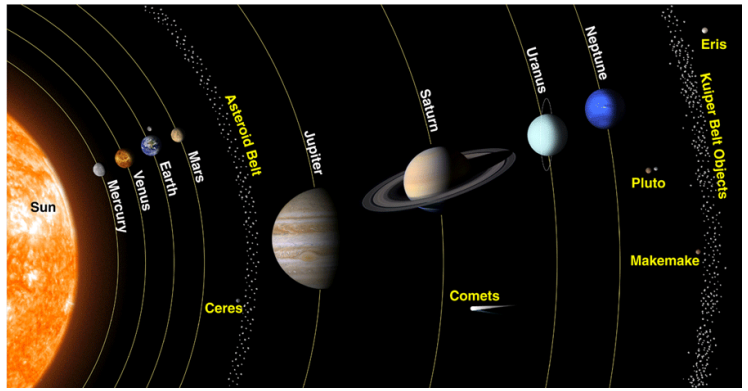


Image source: http://spaceplace.nasa.gov/review/ice-dwarf/solar-system-lrg.png

HANDS ON: COMPILING AND RUNNING N-BODY

nbody.chpl

Things to try

```
chpl nbody.chpl
time ./nbody -nl 1
time ./nbody -nl 1 -n=100000

chpl --fast nbody.chpl
time ./nbody -nl 1
time ./nbody -nl 1 -n=100000
```

```
// number of timesteps to simulate
config const n = 10000;
...
```

Key concepts

- *nix 'time' command is an easy way to see how long a program takes to run
- Compile with '--fast' to have 'chpl' compiler generate faster code

VARIABLES, CONSTANTS, AND OPERATORS

5-BODY IN CHAPEL: VARIABLE AND RECORD DECLARATIONS

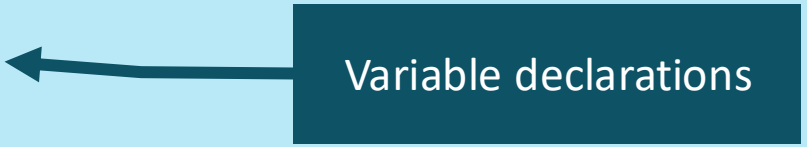
nbody.chpl

```
const pi = 3.141592653589793,
      solarMass = 4 * pi**2,
      daysPerYear = 365.24;

config const numsteps = 10000;

record body {
  var pos: 3*real;
  var v: 3*real;
  var mass: real;
}


…
```



VARIABLES, CONSTANTS, AND PARAMETERS

Basic syntax

```
declaration:
  var   identifier [: type] [= init-expr];
  const identifier [: type] [= init-expr];
  param identifier [: type] [= init-expr];
```

Examples

```
const pi: real = 3.14159;
var count: int;              // initialized to 0
param debug = true;          // inferred to be bool
```

Meaning

- var/const: execution-time variable/constant
- param: compile-time constant
- No init-expr ⇒ initial value is the type's default
- No type ⇒ type is taken from init-expr

PRIMITIVE TYPES

Syntax

Type	Description	Default Value	Currently-Supported Bit Widths	Default Bit Width
bool	logical value	false		impl. dep.
int	signed integer	0	8, 16, 32, 64	64
uint	unsigned integer	0	8, 16, 32, 64	64
real	real floating point	0.0	32, 64	64
imag	imaginary floating point	0.0i	32, 64	64
complex	complex floating points	0.0 + 0.0i	64, 128	128
string	character string	""	N/A	N/A

Examples

```
primitive-type:
   type-name [( bit-width )]
```

```
int(16)   // 16-bit int
real(32)  // 32-bit real
uint      // 64-bit uint
```

CHAPEL'S STATIC TYPE INFERENCE


nbody.chpl

```
const pi = 3.14,              // pi is a real
      coord = 1.2 + 3.4i,     // coord is a complex…
      coord2 = pi*coord,      // …as is coord2
      name = "brad",          // name is a string
      verbose = false;        // verbose is boolean

proc addem(x, y) {            // addem() has generic arguments
  return x + y;               //  and an inferred return type
}

var sum = addem(1, pi),                  // sum is a real
    fullname = addem(name, "ford");      // fullname is a string

writeln((sum, fullname));
```

```
(4.14, bradford)
```

BASIC OPERATORS AND PRECEDENCE

Operator	Description	Associativity	Overloadable
:	cast	left	yes
**	exponentiation	right	yes
! ~	logical and bitwise negation	right	yes
* / %	multiplication, division and modulus	left	yes
(unary) + -	positive identity and negation	right	yes
<< >>	shift left and shift right	left	yes
&	bitwise/logical and	left	yes
^	bitwise/logical xor	left	yes
|	bitwise/logical or	left	yes
+ -	addition and subtraction	left	yes
<= >= < >	ordered comparison	left	yes
== !=	equality comparison	left	yes
&&	short-circuiting logical and	left	via isTrue
||	short-circuiting logical or	left	via isTrue

5-BODY IN CHAPEL: DECLARATIONS

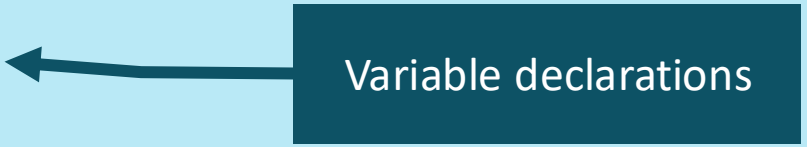
nbody.chpl

```
const pi = 3.141592653589793,
      solarMass = 4 * pi**2,
      daysPerYear = 365.24;

config const numsteps = 10000;

record body {
  var pos: 3*real;
  var v: 3*real;
  var mass: real;
}


…
```



5-BODY IN CHAPEL: DECLARATIONS

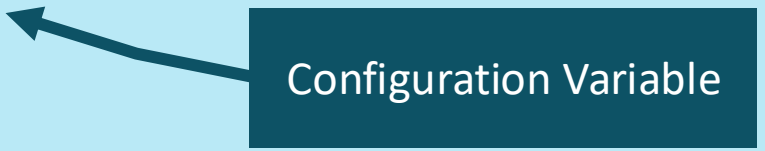
nbody.chpl

```
const pi = 3.141592653589793,
      solarMass = 4 * pi**2,
      daysPerYear = 365.24;

config const numsteps = 10000;

record body {
  var pos: 3*real;
  var v: 3*real;
  var mass: real;
}


…
```



5-BODY IN CHAPEL: DECLARATIONS

nbody.chpl

```
const pi = 3.141592653589793,
      solarMass = 4 * pi**2,
      daysPerYear = 365.24;

config const numsteps = 10000;

record body {
  var pos: 3*real;
  var v: 3*real;
  var mass: real;
}


…
```

Configuration Variable

```
$ ./nbody --numsteps=100
```

CONFIGS

02-configs.chpl

```
param intSize = 32;
type elementType = real(32);
const epsilon = 0.01:elementType;
var start = 1:int(intSize);
```

CONFIGS

02-configs.chpl

```
config param intSize = 32;
config type elementType = real(32);
config const epsilon = 0.01:elementType;
config var start = 1:int(intSize);
```

```
$ chpl 02-configs.chpl -sintSize=64 -selementType=real
$ ./02-configs-start=2 -nl 1 --epsilon=0.00001
```

Experiment some with 02-configs.chpl

1. Which of the above can be changed at compile time?
2. Which can be changed at runtime?

OUTLINE: OVERVIEW OF PROGRAMMING IN CHAPEL

- Running Example: n-body computation (Hands On)
- Variables, Constants, and Operators
- Records and Classes
- Tuples
- Arrays
- Writing out Tuples, Records, and Arrays (Hands On)

5-BODY IN CHAPEL: DECLARATIONS

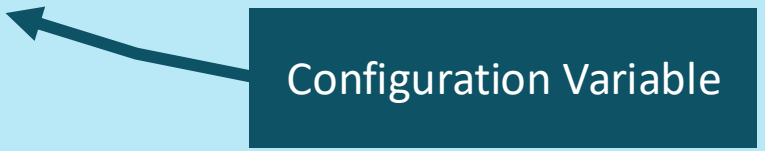
nbody.chpl

```
const pi = 3.141592653589793,
      solarMass = 4 * pi**2,
      daysPerYear = 365.24;

config const numsteps = 10000;

record body {
  var pos: 3*real;
  var v: 3*real;
  var mass: real;
}


…
```



5-BODY IN CHAPEL: DECLARATIONS

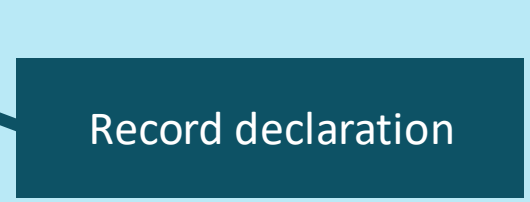
nbody.chpl

```
const pi = 3.141592653589793,
      solarMass = 4 * pi**2,
      daysPerYear = 365.24;

config const numsteps = 10000;

record body {
  var pos: 3*real;
  var v: 3*real;
  var mass: real;
}


…
```



RECORDS AND CLASSES

RECORDS AND CLASSES

02-records-and-classes.chpl

Chapel's object types

- Contain variable definitions (fields)
- Contain procedure & iterator definitions (methods)
- Records: value-based (e.g., assignment copies fields)
- Classes: reference-based (e.g., assignment aliases object)

Example

```
use Math;
record circle {
  var radius: real;
  proc area() {
    return pi*radius**2;
  }
}
```

```
var c1 = new circle(radius=1.0);
var c2 = c1;     // copies c1
c1.radius = 5.0;
writeln(c2.radius);   // prints 1.0
```

RECORDS AND CLASSES

Chapel's object types

- Contain variable definitions (fields)
- Contain procedure & iterator definitions (methods)
- Records: value-based (e.g., assignment copies fields)
- Classes: reference-based (e.g., assignment aliases object)

Example

Experiment some with 02-records-and-classes.chpl

1. What happens if you take away the '?'?
2. What happens if you take away the '!' ?

```
use Math;
class Circle {
  var radius: real;
  proc area() {
    return pi*radius**2;
  }
}
```

```
// c1 is a nilable class
var c1: Circle? = new shared Circle(radius=1.0);
var c2 = c1;              // aliases c1's circle
c1!.radius = 5.0;
writeln(c2!.radius); // prints 5.0
```

CLASSES VS. RECORDS

Classes

- heap-allocated
  - Variables point to objects
  - Support mem. mgmt. policies
- ‘reference' semantics
  - compiler will only copy pointers
- support inheritance
- support dynamic dispatch
- identity matters most
- similar to Java classes

Records

- allocated in-place
  - Variables are the objects
  - Always freed at end of scope
- 'value' semantics
  - compiler may introduce copies
- no inheritance
- no dynamic dispatch
- value matters most
- similar to C++ structs
  - (sans pointers)

5-BODY IN CHAPEL: DECLARATIONS

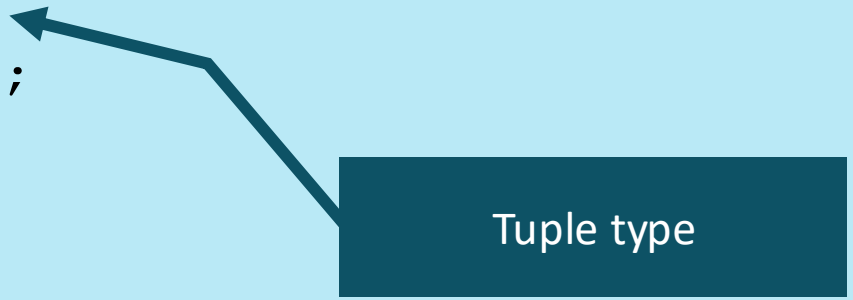
nbody.chpl

```
const pi = 3.141592653589793,
      solarMass = 4 * pi**2,
      daysPerYear = 365.24;

config const numsteps = 10000;

record body {
  var pos: 3*real;
  var v: 3*real;
  var mass: real;
}

…
```



TUPLES

TUPLES

02-tuples.chpl

Use

- support lightweight grouping of values
  - e.g., passing/returning multiple procedure arguments at once
  - short vectors
  - multidimensional array indices
- support heterogeneous data types

Examples

```
var coord: (int, int, int) = (1, 2, 3);
var coordCopy: 3*int = coord;
var (i1, i2, i3) = coord;
var triple: (int, string, real) = (7, "eight", 9.0);
```

5-BODY IN CHAPEL: DECLARATIONS

nbody.chpl

```
const pi = 3.141592653589793,
      solarMass = 4 * pi**2,
      daysPerYear = 365.24;

config const numsteps = 10000;

record body {
  var pos: 3*real;
  var v: 3*real;
  var mass: real;
}

…
```

Variable declarations

Configuration Variable

Record declaration

Tuple type

5-BODY IN CHAPEL: THE BODIES

nbody.chpl

```
var bodies =
    [  /* sun */
       new body(mass = solarMass),

       /* jupiter */
       new body(pos = ( 4.8414314424647209e+00,
                       -1.1603200440274284e+00,
                       -1.0362204447112310e-01),
                  v = ( 1.6600766427440369e-03 * daysPerYear,
                        7.6990111841974042e-03 * daysPerYear,
                       -6.9046001697206302e-05 * daysPerYear),
               mass =   9.5479193842432661e-04 * solarMass),

       /* saturn */
       new body(…),

       /* uranus */
       new body(…),

       /* neptune */
       new body(…)
    ];
```

5-BODY IN CHAPEL: THE BODIES

nbody.chpl

```
var bodies =
    [  /* sun */
       new body(mass = solarMass),

       /* jupiter */
       new body(pos = ( 4.8414314424647209e+00,
                       -1.1603200440274284e+00,
                       -1.0362204447112310e-01),
                  v = ( 1.6600766427440369e-03 * daysPerYear,
                        7.6990111841974042e-03 * daysPerYear,
                       -6.9046001697206302e-05 * daysPerYear),
                mass =   9.5479193842432661e-04 * solarMass),

       /* saturn */
       new body(…),

       /* uranus */
       new body(…),

       /* neptune */
       new body(…)
    ];
```

Create a record object

5-BODY IN CHAPEL: THE BODIES

nbody.chpl

```
var bodies =
    [  /* sun */
       new body(mass = solarMass),

       /* jupiter */
       new body(pos = ( 4.8414314424647209e+00,
                       -1.1603200440274284e+00,
                       -1.0362204447112310e-01),
                  v = ( 1.6600766427440369e-03 * daysPerYear,
                        7.6990111841974042e-03 * daysPerYear,
                       -6.9046001697206302e-05 * daysPerYear),
                mass =   9.5479193842432661e-04 * solarMass),

       /* saturn */
       new body(…),

       /* uranus */
       new body(…),

       /* neptune */
       new body(…)
    ];
```

Tuple values

5-BODY IN CHAPEL: THE BODIES

nbody.chpl

```
var bodies =
    [  /* sun */
       new body(mass = solarMass),

       /* jupiter */
       new body(pos = ( 4.8414314424647209e+00,
                       -1.1603200440274284e+00,
                       -1.0362204447112310e-01),
                  v = ( 1.6600766427440369e-03 * daysPerYear,
                        7.6990111841974042e-03 * daysPerYear,
                       -6.9046001697206302e-05 * daysPerYear),
                mass =   9.5479193842432661e-04 * solarMass),

       /* saturn */
       new body(…),

       /* uranus */
       new body(…),

       /* neptune */
       new body(…)
    ];
```

Array value

ARRAYS

ARRAY TYPES

02-array-examples.chpl

Syntax

```
array-type:
  [ domain-expr ] elt-type
array-value:
  [elt1, elt2, elt3, … eltn]
```

Meaning

- array-type: stores an element of elt-type for each index
- array-value: represent the array with these values

Examples

```
var A: [1..3] int,                  // A stores 0, 0, 0
    B = [5, 3, 9],                  // B stores 5, 3, 9
    C: [1..m, 1..n] real,           // 2D m by n array of reals
    D: [1..m][1..n] real;           // array of arrays of reals
```

More on arrays in data parallelism section later…

5-BODY IN CHAPEL: THE BODIES

nbody.chpl

```
var bodies =
    [  /* sun */
       new body(mass = solarMass),

       /* jupiter */
       new body(pos = ( 4.8414314424647209e+00,
                       -1.1603200440274284e+00,
                       -1.0362204447112310e-01),
                  v = ( 1.6600766427440369e-03 * daysPerYear,
                        7.6990111841974042e-03 * daysPerYear,
                       -6.9046001697206302e-05 * daysPerYear),
                mass =   9.5479193842432661e-04 * solarMass),

       /* saturn */
       new body(…),

       /* uranus */
       new body(…),

       /* neptune */
       new body(…)
    ];
```

Create a record object

Tuple values

Array value

HANDS ON: WRITING TUPLES, RECORDS, AND ARRAYS

Put a 'writeln("bodies = ", bodies);' into program

```
chpl nbody.chpl
./nbody -nl 1
bodies =(pos = (0.0, 0.0, 0.0), vel = (0.0, 0.0, 0.0),
mass = 39.4784) (pos = (4.84143, -1.16032, -0.103622), vel
= (0.606326, 2.81199, -0.0252184), mass = 0.0376937) (pos
= (8.34337, 4.1248, -0.403523), vel = (-1.01077, 1.82566,
0.00841576), mass = 0.0112863) (pos = (12.8944, -15.1112,
-0.223308), vel = (1.08279, 0.868713, -0.0108326), mass =
0.00172372) (pos = (15.3797, -25.9193, 0.179259), vel =
(0.979091, 0.594699, -0.034756), mass = 0.00203369)
-0.169075164
-0.169016441
```