Introduction to C_0

S M Meesum
KREA University

Parts of a program

1. Types
2. Expressions
3. Functions
4. Commands/Statements
5. Libraries

and

6. Contracts

Types

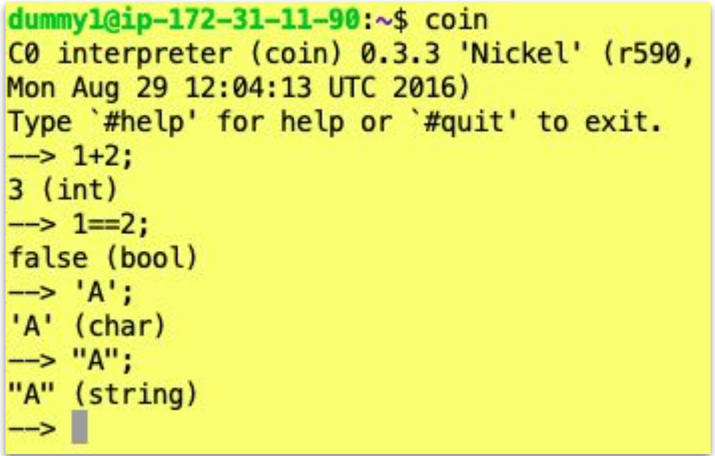
TYPE	DESCRIPTION	VALUES
int	integers	-2^{31} to $2^{31}-1$
char	single character, enclosed in single quotes	ASCII
bool	logical truth values	true, false
string	strings enclosed in double quotes	“hello world!” etc.

And arrays, pointers, structs. More on them later.

Expressions

… can be evaluated.

1. Each expression is of a unique type.
2. Terminated with a semicolon.
   a. 1+2;
   b. 1==2; … etc
3. Constructed using **constants**, **variables**, **operators**, and **function calls**.



```
dummy1@ip-172-31-11-90:~$ coin
C0 interpreter (coin) 0.3.3 'Nickel' (r590,
Mon Aug 29 12:04:13 UTC 2016)
Type `#help' for help or `#quit' to exit.
--> 1+2;
3 (int)
--> 1==2;
false (bool)
--> 'A';
'A' (char)
--> "A";
"A" (string)
-->
```

Constants : parts of expression

… never change

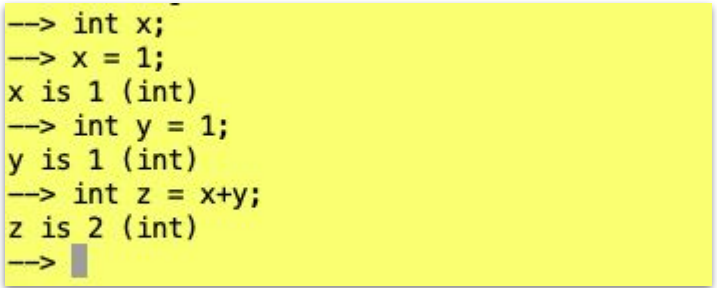
1. Built in constants:
    a. 42
    b. true/false
    c. “Hello!”
    d. ‘E’
2. For each type there may be constants.

a.	42	: int
b.	true/false	: bool
c.	“Hello!”	: string
d.	‘E’	: char

Variables : parts of expression

… store stuff.

1. Variables need to be declared before use.
2. Each variable needs to be declared with its type.
   a. int x = 1;
   b. char c;



```
--> int x;
--> x = 1;
x is 1 (int)
--> int y = 1;
y is 1 (int)
--> int z = x+y;
z is 2 (int)
--> 
```

Operators : parts of expression

… take expression(s) of the same type and return a value.

1. Unary
   a. Minus : -1, -2 …
   b. boolean Not : !false
2. Binary int operators
   a. + (addition), - (subtraction), * (multiplication), / (division), %(modulus)
   b. For comparing ints <, <=, >=, >, ==, != (these return a boolean)
   c. bitwise operators
3. Binary bool operators
   a. Conjunction && (and), disjunction || (or).
   b. Comparisions : ==, !=

Functions

… take things and do something with them.

```
int funcName(int arg1, char arg 2, int arg3) {

        Body of the function

}
```

In general:

T funcName(T_1 x_1, T_2 x_2, …, T_n x_n) {

        Body of the function

}

Functions

```
T funcName(T1 x1, T2 x2, …, Tn xn) {

    Body of the function

}
```

1. T is the return type (T = void, if function returns nothing)
2. T_i is type of argument bound to parameter x_i …
3. The parameters x1, x2, … are local to the body of the function.
4. A function must be declared before it is invoked.
5. A function call, g(e1, e2, …) returns a value of type T. Its arguments are first evaluated in sequence and then bound to the parameters of the function.

Commands/Statements

… are of many types.

1.	Assignment of expression value to variable.	x = e;
2.	Conditionals:	if else
3.	Loops:	while, for
4.	Blocks:	A bunch of statements within { }
5.	Returns:	return e; ***vs*** return;

Arrays

… collecting elements of the same type.

1. The type of an array with elements of type T is denoted T [ ].
2. **alloc_array(T, n)** creates an array of type T with int length n.
3. Index starts from 0 and ends at n-1.
4. An i^th element of array A is A[i-1]. Only 0 to n-1 are valid indices.