CMSC 430: Introduction to Compilers

Dodger: Characters

Announcements

- Assignment 3: Due tomorrow
- Assignment 4: will be available tomorrow.

- Today:
  - Review: C runtime
  - Dodger

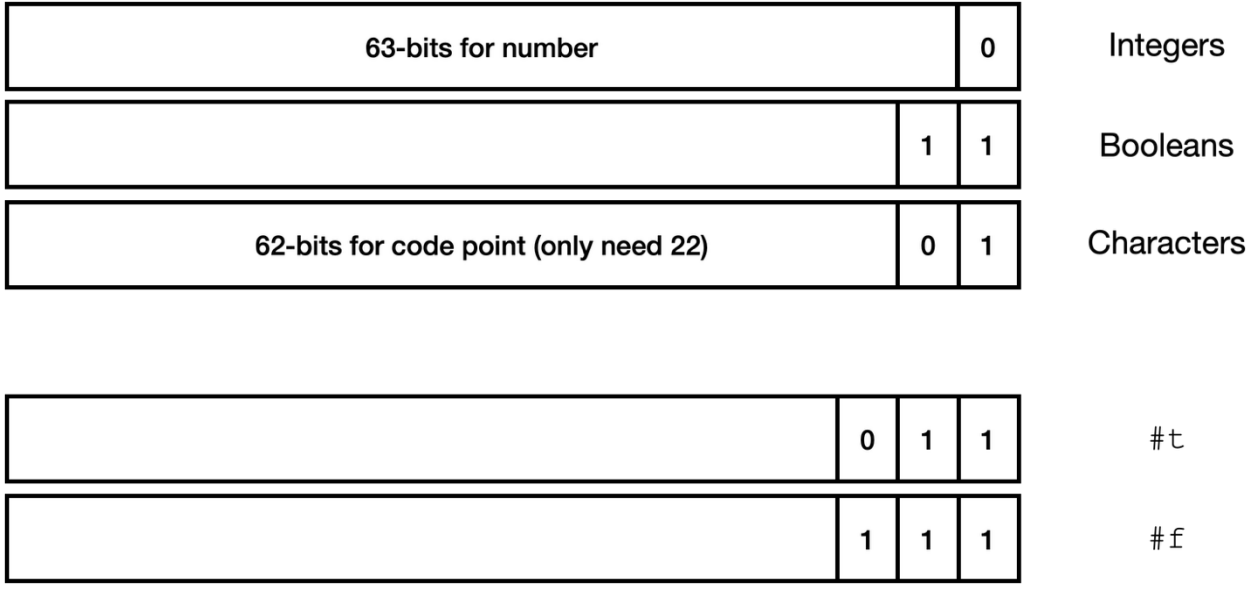
Dodger:

- Dodger adds a new type: character

- It also adds the following operations:
  - char? : Any -> Boolean: predicate for recognizing character values
  - integer->char : Integer -> Character: converts from integers to characters
  - char->integer : Character -> Integer: converts from integers to characters

Encoding values in Dodger

- Type tag in least significant bits



Representing Values with Bits in Dodger

Values	Bits	Decimal
0	0000	0
1	0010	2
2	0100	4
#t	0011	3
#f	0111	7
#\a	1 1000 0101	389
#\b	1 1000 1001	393
#\z	1 1000 1001	489

97*4+1=389

Dodger: Parser

```
(define (datum? x)
      (or (exact-integer? x)
      (boolean? x)
      (char? x)))


(define (op1? x)
  (memq x '(add1 sub1 zero?
          char? integer->char char->integer)))
```

Dodger: Interpreter

```
(define (interp-prim1 op v)
 (match op
  ['add1 (add1 v)]
  ['sub1 (sub1 v)]
  ['zero? (zero? v)]
  ['char? (char? v)]
  ['integer->char (integer->char v)]
  ['char->integer (char->integer v)]))
```

Values -> Integer

```
(define (value->bits v)
(cond
  [(eq? v #t) #b011]
  [(eq? v #f) #b111]
  [(integer? v)
     (arithmetic-shift v int-shift)]
  [(char? v) (bitwise-ior type-char
     (arithmetic-shift (char->integer v)
                       char-shift))]))
```

Dodger Compiler

- We need implement:
  - char?
  - char->integer
  - integer->char

Dodger: Let’s implement it!

CMSC 430: Introduction to Compilers

Evildoer: Calling external functions

Evildoer: Calling external functions

- Evildoer adds a mechanism for interacting with the outside world. It will be able to read and write a byte of information at a time (i.e. an integer between 0 and 256) from the standard input port and output port, respectively.
- Evildoer adds the following operations:
  - write-byte : Byte -> Void: writes given byte to stdout, produces nothing.
  - read-byte : -> Byte or EOF: reads a byte from stdin, if there is one, EOF otherwise.
  - peek-byte : -> Byte or EOF: peeks a byte from stdin, if there is one, EOF otherwise.

Evildoer: Calling external functions

- Evildoer adds the following values:
  - eof : EOF bound to the end-of-file value, and
  - void : -> Void a function that produces the void value.
- adds a predicate that recognizes end-of-file value:
  - eof-object? : Any -> Boolean: determines if argument is the eof value.
- adds a sequence construct:
  - (begin e0 e1): evaluates e0, then e1.

Evildor: Syntax

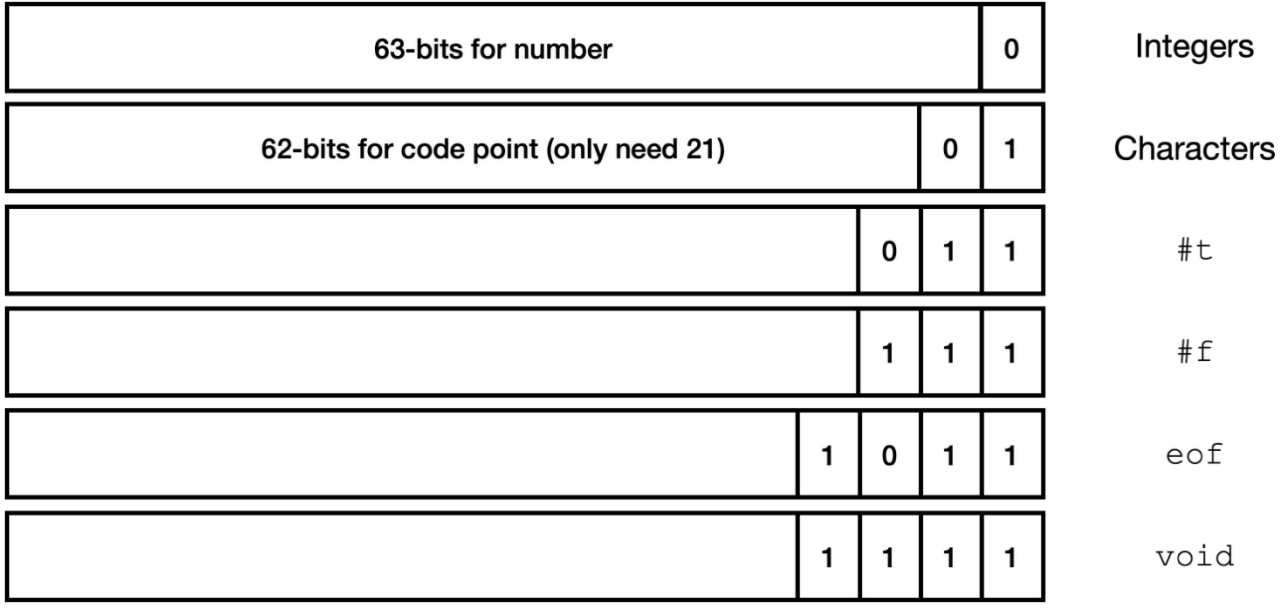
Concrete syntax

```
(begin e1 e2)
(read-byte)
(peek-byte)
(void)
(write-byte e)
(eof-object? e)
```

Abstract syntax

```
(Begin e1 e2)
(Prim0 'read-byte)
(Prim0 'peek-byte)
(Prim0 'void)
(Prim1 'write-byte e)
(Prim1 'eof-object? e)
```

Encoding Values in Evildoer



System V ABI Calling Convention

- The stack must be aligned to a 16-byte boundary when calling a function
- The first six integer or pointer arguments are passed in registers RDI, RSI, RDX, RCX, R8, R9
- Volatile (caller-saved): RAX, RCX, RDX, R8, R9, R10, R11
- nonvolatile (callee-saved): RBX, RBP, RDI, RSI, RSP, R12, R13, R14, and R15

Example

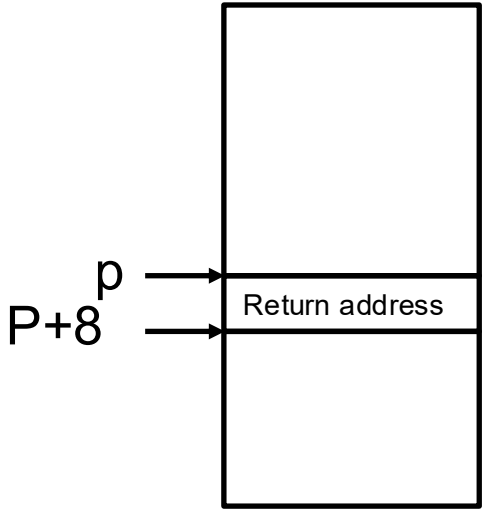
```
#include <stdio.h>
#include <inttypes.h>
int64_t entry();
int main(int argc, char** argv) {
  printf("%" PRId64 "\n", entry());
  return 0;
}
```

```
#include <inttypes.h>
int64_t dbl(int64_t x) {
return x + x;
}
```

```
 default rel
         section .text
         extern _dbl
         global _entry
_entry:
         sub rsp, 8
         mov rdi, 21
         call _dbl
         add rsp, 8
         ret
```

16-byte stack alignment

x86-64 System V ABI requires the stack is 16-byte aligned just before the call instruction is called.




```
Main:
    …
    call entry
    mov …
    …
    ret

entry:
    …
```

P: 16-byte aligned

io.c

```
val_t read_byte(void)
{
  char c = getc(in);
  return (c == EOF) ? val_wrap_eof() : val_wrap_byte(c);
}

val_t peek_byte(void)
{
  char c = getc(in);
  ungetc(c, in);
  return (c == EOF) ? val_wrap_eof() : val_wrap_byte(c);
}

val_t write_byte(val_t c)
{
  putc((char) val_unwrap_int(c), out);
  return val_wrap_void();
}
```