C For Java Programmers

Reid Kleckner

Hello World

```java
/**
 * Multi-line comment.
 */
public class Hello {
  public static void main(String[] args) {
    // Single line comment.
    System.out.println("Hello, World!");

    for (int i = 0; i < args.length; i++) {
      System.out.println("arg " + i + ": " + args[i]);
    }
  }
}
```

```c
/* Multi-line comment. */

#include <stdio.h>

int main(int argc, char **argv) {
  // Single line comments are a C99 feature.
  printf("Hello, World!\n");

  // For-loop variable declaration is also a C99 feature.
  for (int i = 0; i < argc; i++) {
    printf("arg %d: %s\n", i, argv[i]);
  }
}
```

Functions

```c
#include <stdio.h>
// Declarations *must* come before uses.
// Declarations usually go in .h files.
int fib(int n);

int main(void) {
  // Prints '2, 3, 5, 8'.
  printf("%d, %d, %d, %d\n",
         fib(3), fib(4), fib(5), fib(6));
}

int fib(int n) {
  if (n == 0 || n == 1) return n;
  return fib(n - 1) + fib(n - 2);
}
```

Preprocessor

```
#include "ktiming.h"
#ifdef __APPLE__
#include "mach/mach_time.h"
#else
#include <time.h>
#endif
#define KTIMING_CLOCK_ID CLOCK_PROCESS_CPUTIME_ID
clockmark_t ktiming_getmark(void)
{
#ifdef __APPLE__
  uint64_t abs_time = mach_absolute_time();
  // ...
#else
  // ...
  int stat = clock_gettime(KTIMING_CLOCK_ID, &temp);
  if (stat != 0) {
#ifndef __CYGWIN__
    stat = clock_gettime(CLOCK_MONOTONIC , &temp);
#endif
    // ...
  }
  // ...
#endif
}
```

Integer Types

- Include <stdint.h> to get exact precision types (a C99 feature)

Type	Precision	Type	Precision
char	~ 8 bit	int8_t	8 bit
short	~ 16 bit	int16_t	16 bit
int	~ 32 bit	int32_t	32 bit
long	~ 32 or 64 bit	int64_t	64 bit
long long	> 64 bit	intptr_t	fits pointers
		size_t	fits sizes

Signed and unsigned types

- Can use “unsigned type” or uintN_t variants
- Right shift of signed types uses sign extension
  - ((int8_t)0xFF) >> 4 == 0xFF
  - ((uint8_t)0xFF) >> 4 == 0x0F
- Overflow of signed types is undefined!
  - Compiler will optimize branch away if x is signed:

    if (x + 1 < x) { /* handle overflow */ }
- Use the right integer types

Arrays

```
// Can only declare fixed-size global and
// stack arrays.
int A[5];
int main(void) {
  int B[5];
  A[ 0] = -1;  // OK
  // No bounds checking or exceptions!
  B[-1] = -1;  // BAD!  May work anyway.
  B[ 6] = -1;  // BAD!  May work anyway.
}
```

Simple Pointers

```c
#include <stdio.h>
int main(void) {
  int a = 0;
  int *p = &a; // Takes address of a.
  int* q = p; // Copies p into q.
  printf("p: %p *p: %d\n", p, *p);
  *p = 1;
  printf("a: %d *q: %d\n", a, *q);
  *q = 2;
  printf("a: %d *p: %d\n", a, *p);
}
// Output:
// p: 0x7fff5fbff4dc *p: 0
// a: 1 *q: 1
// a: 2 *p: 2
```

Pointer Arithmetic

```c
#include <stdio.h>
int A[5];
int main(void) {
  for (int i = 0; i < 5; i++)
    A[i] = i;
  // Take addr of elem 0 in A.
  int *p = &A[0];
  // 'p + i' gets the pointer
  // i "slots" over from p
  printf("%d %d %d %d %d\n",
         *(p + 0), *(p + 1), *(p + 2),
         *(p + 3), *(p + 4));
}
// Output:
// 0 1 2 3 4
```

Pointer Pseudo-arrays

```
#include <stdio.h>
int A[5];
int main(void) {
  for (int i = 0; i < 5; i++)
    A[i] = i;
  // Take addr of elem 0 in A.
  int *p = &A[0];
  // "p[i]" means "*(p + i)"
  printf("%d %d %d %d %d\n",
         p[0], p[1], p[2],
         p[3], p[4]);
}
// Output:
// 0 1 2 3 4
```

Memory Allocation

```c
#include <stdlib.h>
int main(int argc, char **argv) {
  int *p = (int*) malloc(
      10 * sizeof(int));
  // p is a 10 element "array" of ints.
  if (argc > 1) {
    return 1;  // Memory leak!
    // Valgrind can find these.
  }
  free(p);
}
```

Aside: Memory Model

- Each process gets a big bucket of (virtual) memory
- Like a giant array of bytes
- Divided (rougly) into Stack, Globals, Heap
- Pointers are indexes into the giant array
- Only some regions are valid, kernel protects rest, bad access → segmentation fault

Structs

```c
#include <stdio.h>
typedef struct Point {
  int x;
  int y;
} Point;
int main(void) {
  Point p;  // Stack-allocate a struct.
  // Use '.' for direct struct access.
  p.x = 5; p.y = 9;
  Point *q = &p;
  // Use '->' for access through pointers.
  printf("x: %d, y: %d\n", q->x, q->y);
}
// Output: x: 5, y: 9
```

Point Example in Java

```
class Point {
  public int x;
  public int y;
  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
  public Point add(Point b) {
    return new Point(this.x + b.x, this.y + b.y);
  }
  public static void main(String[] args) {
    Point a = new Point(1, 2);
    Point b = new Point(3, 4);
    Point c = a.add(b);
    System.out.println("x: " + c.x +
                       ", y: " + c.y);
    // Garbage collection frees a, b, and c.
  }
}
```

Point Example in C

```c
#include <stdlib.h>
#include <stdio.h>
typedef struct Point {
  int x;
  int y;
} Point;
Point *point_new(int x, int y) {
  Point *result = (Point *)malloc(sizeof(Point));
  result->x = x;
  result->y = y;
  return result;
}
Point *point_add(Point *a, Point *b) {
  return point_new(a->x + b->x, a->y + b->y);
}
int main(void) {
  Point *a = point_new(1, 2);
  Point *b = point_new(3, 4);
  Point *c = point_add(a, b);
  printf("x: %d, y: %d\n", c->x, c->y);
  free(a); free(b); free(c);
}
```

C Family History

- Born 1969-1973 at Bell Labs, Brian Kernighan and Dennis Ritchie (K&R C)
- “C with Classes” Bjarne Stroustroup in 1979, became C++ in 1983, added many features
- Straight C standardized by ANSI in 1989 (C89 or ANSI C)
- Straight C standardized again in 1999 (C99), borrowing more C++ features

6.172 Performance Engineering of Software Systems

Fall 2010