COMP 2355 Introduction to Systems Programming

Christian Grothoff
christian@grothoff.org

http://grothoff.org/christian/

Functions are Values

```
int f() {
  printf("Having fun?");
  return 4;
}
int main(int argc, char ** argv) {
  void * ptr = &f;
  return 0;
}
```

Functions have Types

```
typedef int (*Fun)(void);
int f() {
  printf("Having fun?");
  return 4;
}
int main(int argc, char ** argv) {
  Fun fptr = &f;
  int (*fptr2)(void) = &f;
  return 0;
}
```

Functions have Types

```
typedef int (*Fun)(float, const char*, ...);
typedef int (*Fun2)(float f,const char* c, ...);
int f(float a, const char * s, ...) {
  printf("Having fun?");
  return 4;
}
int main(int argc, char ** argv) {
  Fun fptr = &f;
  Fun2 fptr = &f;
  return 0;
}
```

Function Values can be called!

```
typedef int (*Fun)(int);
int f(int val) {
  printf("Having fun?");
  return val + 1;
}
int main(int argc, char ** argv) {
  Fun fptr = &f;
  fptr(4);
  return 0;
}
```

A Simple Example

```
typedef double (*MathFun)(double);
double fixpoint(MathFun func, double curr) {
  double last;

  last = curr + 1.0;
  while (last != curr) {
    last = curr;
    curr = func(last);
  }
  return last;
}
```

Questions

?

Question!

How would you implement this kind of function in Java?

Sorting

```
typedef int (*Compare)(const void * a1, const void * a2);

void qsort(void *base, size_t nmemb, size_t size,
           Compare comparator);

Compare scmp = casesensitive ? &strcasecmp : &strcmp;
```

Searching

```
typedef int (*Compare)(const void * a1, const void * a2);

void * bsearch(const void *key, const void *base,
               size_t nmemb, size_t size,
               Compare comparator);
```

Functions in Structs

```
struct Funny {
  int val;
  int (*fun)(int);
};
static int run(struct Funny * f) {
  f->val = f->fun(f->val);
}
```

Functions in Structs

```
static int add_one(int arg) {
  return arg + 1;
}
int main(int argc, char ** argv) {
  struct Funny f;
  f.val = 42;
  f.fun = &add_one;
  run(&f);
  printf("%d\n", f.val);
  return 0;
}
```

Classes in C: Declaration

```
struct Point2D {
  int x; int y;
  void (*translate)(struct Point2D * self,
                    int dx, int dy);
  double (*distance)(const struct Point2D * self,
                     const struct Point2D * other);
  void (*draw)(struct Point2D * self, ...);
};
```

Classes in C: Constructor

```
struct Point2D * point_create(int x, int y) {
  struct Point2D * ret;
  ret = malloc(sizeof(struct Point2D));
  ret->x = x;
  ret->y = y;
  ret->translate = &point_translate;
  ret->distance = &point_distance;
  ret->draw = &point_draw;
  return ret;
}
```

Classes in C: Methods

```
static void point_translate(struct Point2D * self,
                            int dx, int dy) {
  self->x += x;
  self->y += y;
}
#define SQUARE(a) ((a)*(a))
static double
point_distance(const struct Point2D * self,
               const struct Point2D * other) {
  return sqrt(SQUARE(self->x - other->x) +
              SQUARE(self->y - other->y));
}
```

Classes in C: Use

```
int main(int argc, char ** argv) {
  struct Point2D * p1;
  struct Point2D * p2;
  p1 = point_create(2,4);
  p2 = point_create(4,2);
  p1->translate(p1, 2, 2);
  printf("Distance: %f\n",
         p1->distance(p1, p2));
};
```

Classes in C: Inheritance

```
struct ColorPoint2D {
  struct Point2D parent;
  int r, g, b;
};
```

Classes in C: Overriding

```
struct ColorPoint2D *
color_point_create(int x, int y,
                   int r, int g, int b) {
  struct ColorPoint2D * ret;
  ret = realloc(point_create(x, y),
                sizeof(struct ColorPoint2D));
  ret->r = r;
  ret->g = g;
  ret->b = b;
  ret->parent.draw = &color_point_draw;
  return ret;
}
```

Classes in C: Overriding Methods

```
static void
color_point_draw(struct Point2D * self,
                 ...) {
  struct ColorPoint2D * real_self;
  real_self = (struct ColorPoint2D *) self;
  // ...
}
```

Fun Fact

Early versions of C++ compilers used virtually the same approach to add classes and objects to C.

The first C++ compilers simply translated C++ source code to C source code.

GNU Extension: Inner Functions

```
void process(int (*generator)(void)) {
  printf("%d\n", generator());
  printf("%d\n", generator());
  printf("%d\n", generator());
}
int outer() {
  int i = 0;
  int inner() { return i++;  }
  process(&inner);
  return i;
}
```

Important Fact

The GNU C compiler implements inner functions using a trick called "trampolines".

Trampolines require the stack to be executable, which is a security risk.

Alternatives

```
void process(int (*generator)(void *), void * cls) {
  printf("%d\n", generator(cls));
  printf("%d\n", generator(cls));
  printf("%d\n", generator(cls)); }
static int former_inner(void * cls) {
  int * iptr = cls;
  return (*iptr)++;  }
int outer() {
  int i = 0;
  process(&former_inner, &i);
  return i;  }
```

A Warning

```
typedef int (*Generator)(void);
Generator create_generator() {
  int i = 0;
  int inner() { return i++; }
  return &inner;              }
int main(int argc, char ** argv) {
  Generator g = create_generator();
  printf("%d\n", g());
  printf("%d\n", g());
  printf("%d\n", g());
  return 0; }
```

Questions

?

Question!

How would you pass multiple arguments?