CSE216 Programming Abstractions

C Structure

YoungMin Kwon

Data Abstraction using C struct

- Data abstraction
  - complex_t is a structure for both the rectangular form and the polar form
  - The first parts of rect_t and polar_t should share the same structure of complex_t
  - However, they have different tags, and the function pointers have different values

```
#ifndef __COMPLEX__
#define __COMPLEX__

typedef enum {
    COMPLEX_RECT,
    COMPLEX_POLAR
} complex_tag;

typedef struct complex {
    complex_tag tag;
    double (*real)   (struct complex* self);
    double (*imag)   (struct complex* self);
    double (*mag)    (struct complex* self);
    double (*ang)    (struct complex* self);
    char  *(*to_str) (struct complex* self);
    void   (*destroy)(struct complex* self);
} complex_t;

extern complex_t *make_rect (double r, double i);
extern complex_t *make_polar(double m, double a);
extern complex_t *cmplx_add(complex_t *a, complex_t *b);
extern complex_t *cmplx_sub(complex_t *a, complex_t *b);
extern complex_t *cmplx_mul(complex_t *a, complex_t *b);
extern complex_t *cmplx_div(complex_t *a, complex_t *b);

#endif
```

Complex Numbers

- `rect_t` is a complex number in the rectangular form
  - E.g. 2 + 3i
  - Implement TODOs in rect.c
    - `mag: sqrt(r*r + i*i)`
    - `ang: atan2(i, r)`

```
#ifndef __RECT__
#define __RECT__

#include "complex.h"

typedef struct rect {
    complex_tag tag;
    double (*real)   (complex_t* self);
    double (*imag)   (complex_t* self);
    double (*mag)    (complex_t* self);
    double (*ang)    (complex_t* self);
    char  *(*to_str) (complex_t* self);
    void   (*destroy)(complex_t* self);
    double r;
    double i;
} rect_t;

#endif
```

```c
#include <math.h>
#include <stdlib.h>

#include "common.h"
#include "rect.h"

static rect_t *to_rect(complex_t *self) {
    ON_FALSE_EXIT(self->tag == COMPLEX_RECT, "not a rect type");
    return (rect_t*) self;
}
static double real(complex_t *self) {
    //TODO: impelement this function
}
static double imag(complex_t *self) {
    //TODO: impelement this function
}
static double mag(complex_t *self) {
    //TODO: impelement this function
}
static double ang(complex_t *self) {
    //TODO: impelement this function
}
```

```
static char* to_str(complex_t *self) {
    static char str[100];
    double r = to_rect(self)->r;
    double i = to_rect(self)->i;
    sprintf(str, "%lf + i %lf", r, i);
    return str;
}
static void destroy(complex_t *self) {
    free(self);
}
complex_t *make_rect(double r, double i) {
    rect_t *rect  = malloc(sizeof(rect_t));
    //TODO: construct rect

    return (complex_t*) rect;
}
```

Complex Numbers

- polar_t is a complex number in the polar form
  - E.g. $2\,e^{i\pi}$
  - Implement TODOs in polar.c
    - `real: m * cos(a)`
    - `imag: m * sin(a)`

```
#ifndef __POLAR__
#define __POLAR__

#include "complex.h"

typedef struct polar {
    complex_tag tag;
    double (*real)   (complex_t* self);
    double (*imag)   (complex_t* self);
    double (*mag)    (complex_t* self);
    double (*ang)    (complex_t* self);
    char  *(*to_str) (complex_t* self);
    void   (*destroy)(complex_t* self);
    double m;
    double a;
} polar_t;

#endif
```

```c
#include <math.h>
#include "common.h"
#include "polar.h"

static polar_t *to_polar(complex_t *self) {
    ON_FALSE_EXIT(self->tag == COMPLEX_POLAR, "not a polar type");
    return (polar_t*) self;
}
static double real(complex_t *self) {
    //TODO: impelement this function
}
static double imag(complex_t *self) {
    //TODO: impelement this function
}
static double mag(complex_t *self) {
    //TODO: impelement this function
}
static double ang(complex_t *self) {
    //TODO: impelement this function
}
```

```
static char* to_str(complex_t *self) {
    static char str[100];
    double m = to_polar(self)->m;
    double a = to_polar(self)->a;
    sprintf(str, "%lf e^i %lf", m, a);
    return str;
}
static void destroy(complex_t *self) {
    //TODO: implement this function
}
complex_t *make_polar(double m, double a) {
    //TODO: implement this function
}
```

Complex Numbers

- Implement arithmetic functions for complex numbers
  - Implement TODOs in arith.c
  - Data abstraction
    - We do not need to know whether a complex_t type variable x is implemented as rect_t or as polar_t

```c
#include "complex.h"

complex_t *cmplx_add(complex_t *a, complex_t *b) {
    //TODO: implement this function using make_rect
}
complex_t *cmplx_sub(complex_t *a, complex_t *b) {
    //TODO: implement this function using make_rect
}
complex_t *cmplx_mul(complex_t *a, complex_t *b) {
    //TODO: implement this function using make_polar
}
complex_t *cmplx_div(complex_t *a, complex_t *b) {
    //TODO: implement this function using make_polar
}
```