Structures & ADTs

P. P. Chakrabarti

Some ways to define structures & types:

```
struct {
 float real;
 float imag;
} a [10], c;
```

```
struct  comp_rec {
 float real;
 float imag;
} ;
struct comp_rec a[10],c;
```

```
typedef  struct {
 float real;
 float imag;
}  complex;
complex a[10],c;
```

```
 struct stud_rec{
 char name[20];
 int roll;
 float SGPA[8];
 float CGPA;
} ;
struct stud_rec student[10];
```

```
 typedef struct {
 char name[20];
 int roll;
 float SGPA[8];
 float CGPA;
} record;
record student[10];
```

```
 typedef struct {
 char name[20];
 int roll;
 float SGPA[8];
 float CGPA;
} person[10];
person student;
```

variables

sizeof( )

```
main()
{  struct  comp_rec { float real; float imag; } ;
   struct comp_rec a[10],c;
   typedef struct{
   char name[20];  int roll; float SGPA[8], CGPA;
  } record, person[10];
  person student;
  printf("%d\n", sizeof(struct comp_rec));
  printf("%d\n", sizeof(c));
  printf("%d\n", sizeof(a));
  printf("%d\n", sizeof(record));
  printf("%d\n", sizeof(person));
  printf("%d\n", sizeof(student[3]));
}
```

```
[ppchak]$ ./a.out
8
8
80
60
600
60
```

Abstract Data Types:

```
 typedef struct
{ float real;
  float imag;
} complex;

complex add_c (complex, complex);
complex get_data_c();
void display_c(complex);

main()
{
 complex a,b,c;
 a = get_data_c();
 b = get_data_c();
 c = add_c(a,b);
 display_c(c);
}
```

```
 complex add_c(complex x, complex y)
{
  complex z;
   z.real = x.real + y.real;
   z.imag = x.imag + y.imag;
   return z;
}

complex get_data_c()
{
  complex x;
  scanf("%f%f", &x.real, &x.imag);
  return x;
}

void display_c(complex x)
{
 printf("%f + i %f\n", x.real, x.imag);
}
```

ADT Examples

- Complex number
  - Operations: add, subtract, multiply, divide
- Rational number:
  - Operations: reduce, add, subtract, multiply, divide
- Set
  - Operations: MakeSet, adjoin, member, union, intersection, difference, remove
- Integers of large size
  - Operations: add, subtract, multiply, divide
- Matrices

ADT: Rational Number

Interface functions

Constructor function:

```
RATIONAL makerational (int, int);
```

Selector functions :

```
int numerator (RATIONAL);
Int denominator (RATIONAL) ;
```

Operations:

```
RATIONAL add (RATIONAL,RATIONAL);
RATIONAL mult (RATIONAL, RATIONAL);
RATIONAL reduce (RATIONAL) ;
```

Equality testing :

```
int equal (RATIONAL, RATIONAL);
```

Print :

```
void printrat (RATIONAL) ;
```

ADT: Rational Number

Concrete implementation I

```
typedef struct {
   int numerator;
   int denominator;
}RATIONAL;
```

```
RATIONAL makerational (int x, int y) {
          RATIONAL r;
          r.numerator = x;
          r.denominator = y;
          return r;
}
```

```
int numerator (RATIONAL r)        {
          return r.numerator;
}
int denominator (RATIONAL r)   {
          return r.denominator;
}
```

```
RATIONAL reduce (RATIONAL r) {
     int g;
     g = gcd (r.numerator,r.denominator);
     r.numerator /= g;
     r.denominator /=  g;
     return r;
}
```

ADT: Rational Number

implementation of add (1)

```
typedef struct {
    int numerator;
    int denominator;
} RATIONAL;
```

```
RATIONAL add (RATIONAL r1, RATIONAL r2) {
        RATIONAL r;
        int g;
        g = gcd(r1.denominator,r2.denominator);
        r.denominator = lcm(r1.denominator,r2.denominator);
        r.numerator = r1.denominator*r2.denominator/g;
        r.numerator += r2.denominator*r1.numerator/g;
        return r;
}
```

ADT: Rational Number

implementation of add (2)

```
typedef struct {
    int numerator;
    int denominator;
}RATIONAL;
```

```
RATIONAL add (RATIONAL r1, RATIONAL r2) {
        RATIONAL r;
        r.numerator = r1.numerator*r2.denominator
                +r2.numerator*r1.denominator;
        r.denominator=r1.denominator*r2.denominator;
        return r;
}
```

ADT: Rational Number

Concrete implementation I

```
typedef struct {
    int numerator;
    int denominator;
}RATIONAL;
```

```
RATIONAL mult (ARTIONAL r1, ARTIONAL r2) {
      RATIONAL r;
      r.numerator = r1.numerator*r2.numerator;
      r.denominator = r1.denominator*r2.denominator
      r = reduce (r);
      return r;
}
```

```
nt equal (RATIONAL r1, RATIONAL r2)   {
          return
            (r1.numerator*r2.denominator==r2.numerator*r1.denominator);
```

```
void printrat (RATIONAL r) {
            printf (“%d / %d “,r.numerator, r.denominator);
}
```

ADT: Rational Number

Alternate Concrete implementation II

```
typedef struct {
    int ar[2];
}RATIONAL;
```

ADT: Rational Number

Concrete implementation II

```
typedef struct {
   int ar[2] ;
}RATIONAL;
```

```
RATIONAL makerational (int x, int y) {
        RATIONAL r;
        r.ar[0] = x;
        r.ar[1] = y;
        return r;
}
```

```
int numerator (RATIONAL r)          {
        return r.ar[0];
}
int denominator (RATIONAL r)   {
        return r.ar[1];
}
```

```
RATIONAL reduce (RATIONAL r) {
    int g;
    g = gcd (r.ar[0], r.ar[1]);
    r.ar[0]  /= g;
    r.ar[1] /=  g;
    return r;
}
```

ADT Examples

- Complex number:
  - Operations: add, subtract, multiply, divide
- Rational number:
  - Operations: reduce, add, subtract, multiply, divide
- Set
  - Operations: MakeSet, adjoin, member, union, intersection, difference, remove
- Integers of large size
  - Operations: add, subtract, multiply, divide
- Martices

Pointers to records:

```
main()
{ typedef struct
   { float real;
      float imag;
    } complex;
 complex a;
 complex *t;
 a.real = 5.1;
 a.imag = 8.3;
 t = &a;
 printf("a.real = %.2f\n", t->real);
 printf("a.imag = %.2f\n", t->imag);
 }
```

```
[ppchak]$ ./a.out
a.real = 5.10
a.imag = 8.30
```

More on Pointers to structs:

```
 main()
{  struct { float real; float imag; } a[10], c, *temp;
 int i,n;
 scanf("%d",&n);
 for (i=0; i<n; i++) scanf("%f%f", &a[i].real, &a[i].imag);
 temp = a;
 c.real = 0;  c.imag = 0;
 for (i=0; i<n; i++)
 { c.real += temp[i].real;
    c.imag += (temp+i)->imag;
 }
 for (i=0; i<n; i++)
 printf("a[%d] = %.2f +i %.2f\n", i, a[i].real, a[i].imag);
 printf(" c = %.2f+ i %.2f \n", c.real, c.imag);
}
```

```
[ppchak]$ ./a.out
4
2.3 4.5
1.7 8.9
2.1 3.1
4.5 6.1
a[0] = 2.30 +i 4.50
a[1] = 1.70 +i 8.90
a[2] = 2.10 +i 3.10
a[3] = 4.50 +i 6.10
c = 10.60+ i 22.60
```

Parting Note: :

You can define nested structs but be careful about circular definitions

```
struct my_rec{
char name[20];
struct my_rec x;
float m[8];
 } ;
```

This is an unbounded wrong definition

```
struct my_rec{
char name[20];
struct my_rec *y;
float m[8];
 } ;
```

This is Okay syntactically