Programming

Structures

Example - Complex Numbers

- Structure declaration

```
typedef struct complex {
    double real, img;
} Complex ;
```

- Variable definition

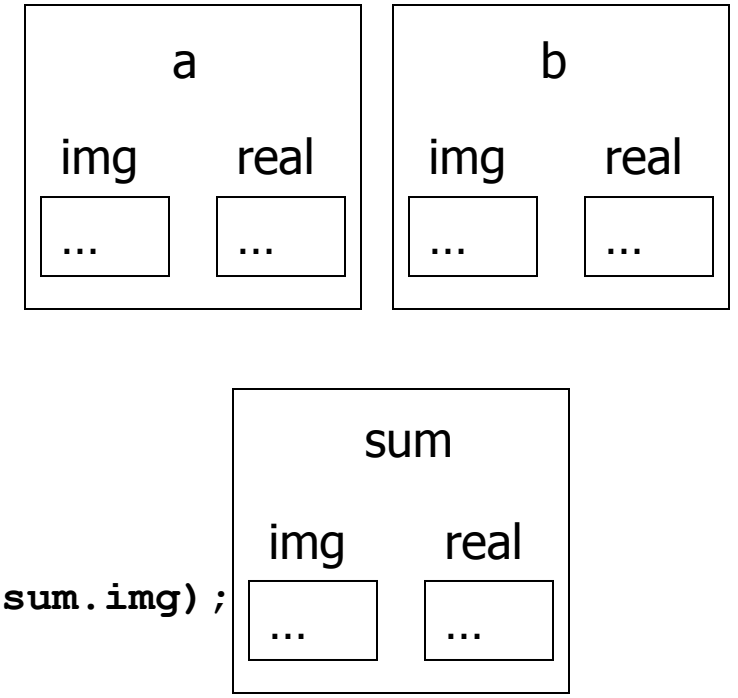
```
Complex c1, c2;
```

Structures and Functions - Example

```
Complex make_complex(double real, double img)
{
    Complex temp;

    temp.real = real;
    temp.img  = img;

    return temp;
}


Complex add_complex(Complex c1, Complex c2)
{
    return make_complex(c1.real + c2.real,
                        c1.img + c2.img);
}
```

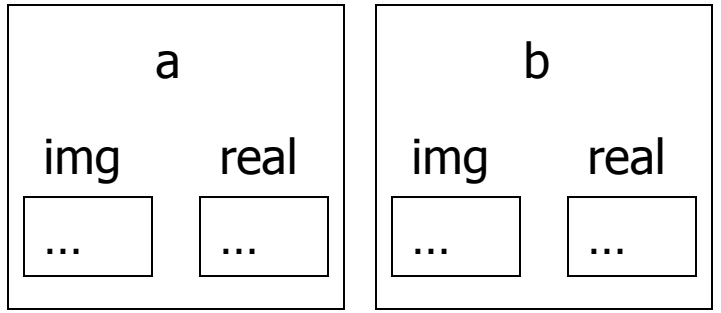
add_complex – step by step

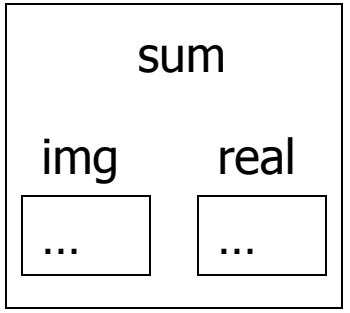
```
int main(void)
{
    Complex a, b, sum;

    printf("…");
    scanf("%lf%lf", &(a.real), &(a.img));
    printf("…");
    scanf("%lf%lf", &(b.real), &(b.img));

    sum = add_complex(a, b);

    printf("result = %g+%gi\n", sum.real, sum.img);

    return 0;
}
```



add_complex – step by step

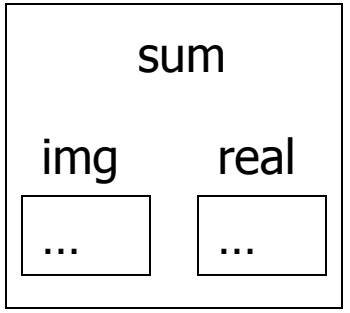
```
int main(void)
{
    Complex a, b, sum;

    printf("…");
    scanf("%lf%lf", &(a.real), &(a.img));
    printf("…");
    scanf("%lf%lf", &(b.real), &(b.img));

    sum = add_complex(a, b);

    printf("result = %g+%gi\n", sum.real, sum.img);

    return 0;
}
```





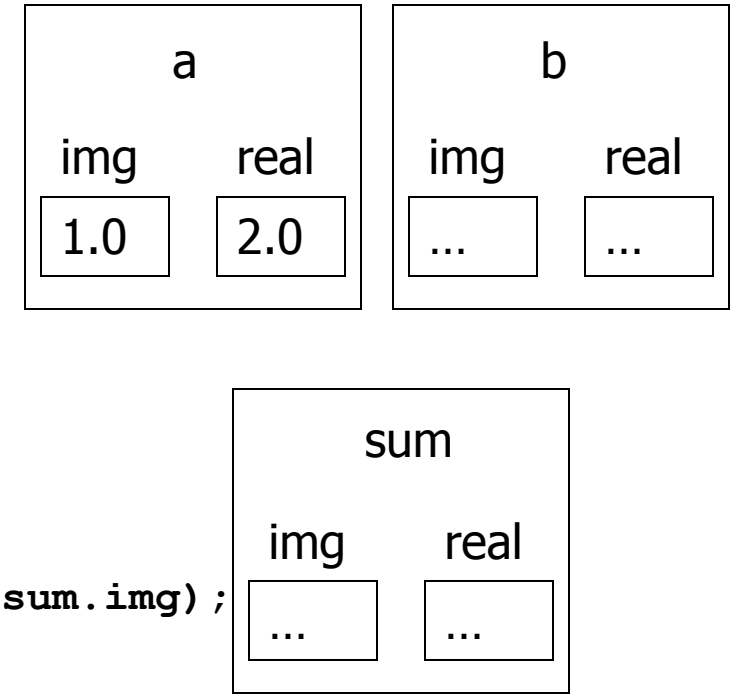
add_complex – step by step

```
int main(void)
{
    Complex a, b, sum;

    printf("…");
    scanf("%lf%lf", &(a.real), &(a.img));
    printf("…");
    scanf("%lf%lf", &(b.real), &(b.img));

    sum = add_complex(a, b);

    printf("result = %g+%gi\n", sum.real, sum.img);

    return 0;
}
```



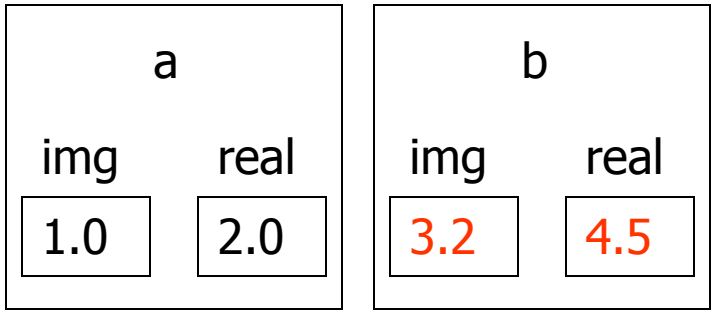
add_complex – step by step

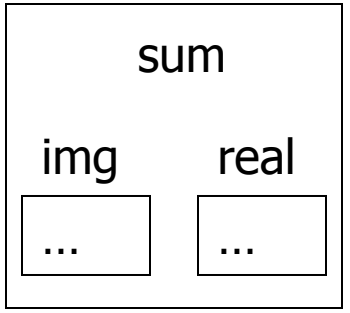
```
int main(void)
{
    Complex a, b, sum;

    printf("…");
    scanf("%lf%lf", &(a.real), &(a.img));
    printf("…");
    scanf("%lf%lf", &(b.real), &(b.img));

    sum = add_complex(a, b);

    printf("result = %g+%gi\n", sum.real, sum.img);

    return 0;
}
```



add_complex – step by step

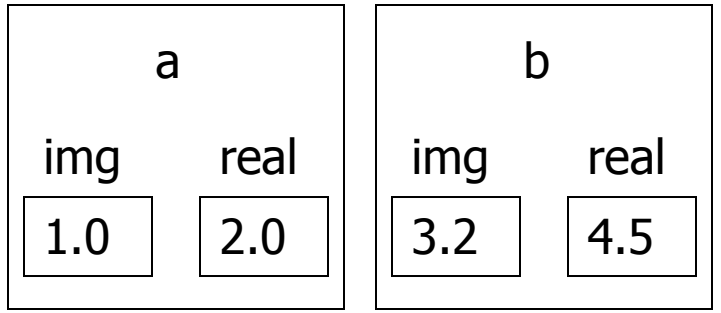
```
int main(void)
{
    Complex a, b, sum;

    printf("…");
    scanf("%lf%lf", &(a.real), &(a.img));
    printf("…");
    scanf("%lf%lf", &(b.real), &(b.img));

    sum = add_complex(a, b);

    printf("result = %g+%gi\n", sum.real, sum.img);

    return 0;
}
```

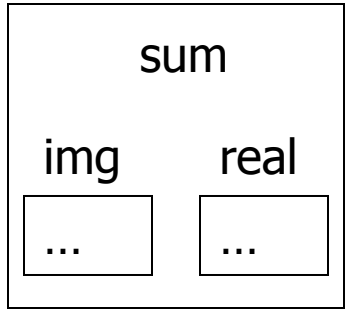




add_complex – step by step

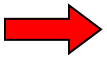
```
int main(void)
{
    Complex a, b, sum;

    printf("…");
    scanf("%lf%lf", &(a.real), &(a.img));
    printf("…");
    scanf("%lf%lf", &(b.real), &(b.img));

    sum = add_complex(a, b);

    printf("result = %g+%gi\n", sum.real, sum.img);

    return 0;
}
```

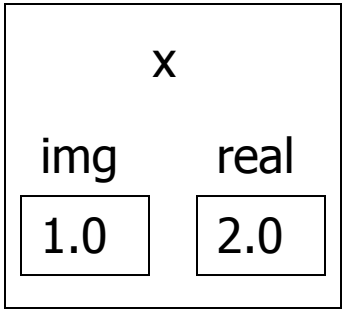


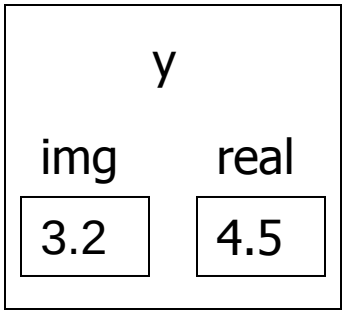


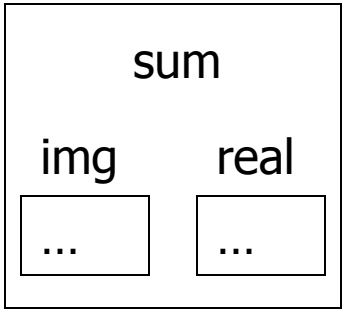
add_complex – step by step

```
Complex add_complex(Complex x, Complex y)
{
    Complex sum;

    sum.real = x.real + y.real;
    sum.img = x.img + y.img;

    return sum;
}
```



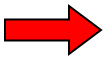


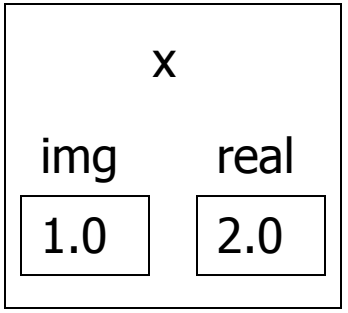


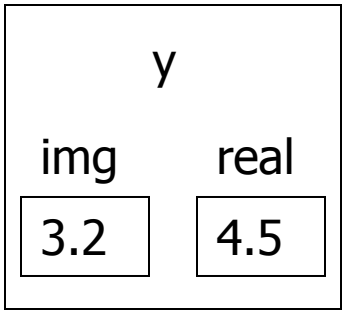


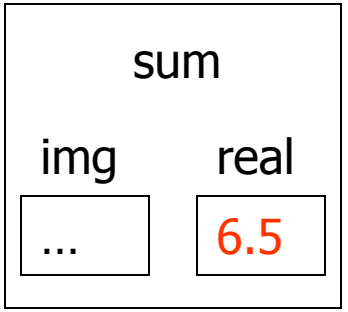
add_complex – step by step

```
Complex add_complex(Complex x, Complex y)
{
    Complex sum;

    sum.real = x.real + y.real;
    sum.img = x.img + y.img;

    return sum;
}
```



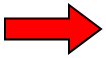


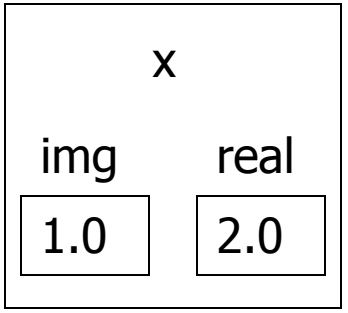


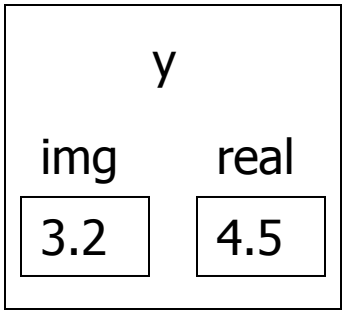


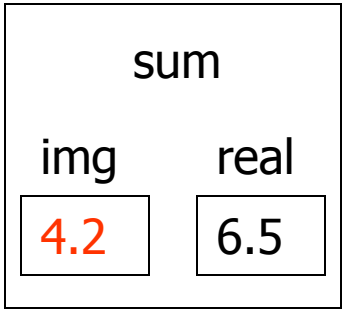
add_complex – step by step

```
Complex add_complex(Complex x, Complex y)
{
    Complex sum;

    sum.real = x.real + y.real;
    sum.img = x.img + y.img;

    return sum;
}
```



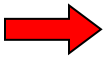


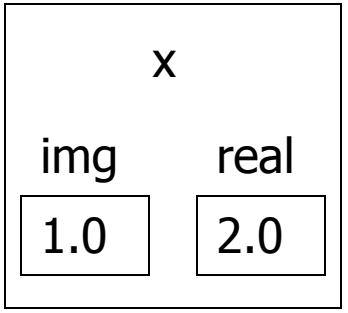


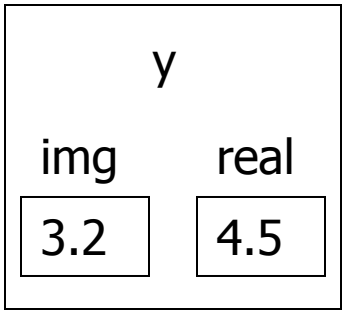


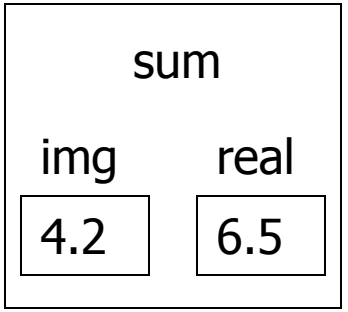
add_complex – step by step

```
Complex add_complex(Complex x, Complex y)
{
    Complex sum;

    sum.real = x.real + y.real;
    sum.img = x.img + y.img;

    return sum;
}
```



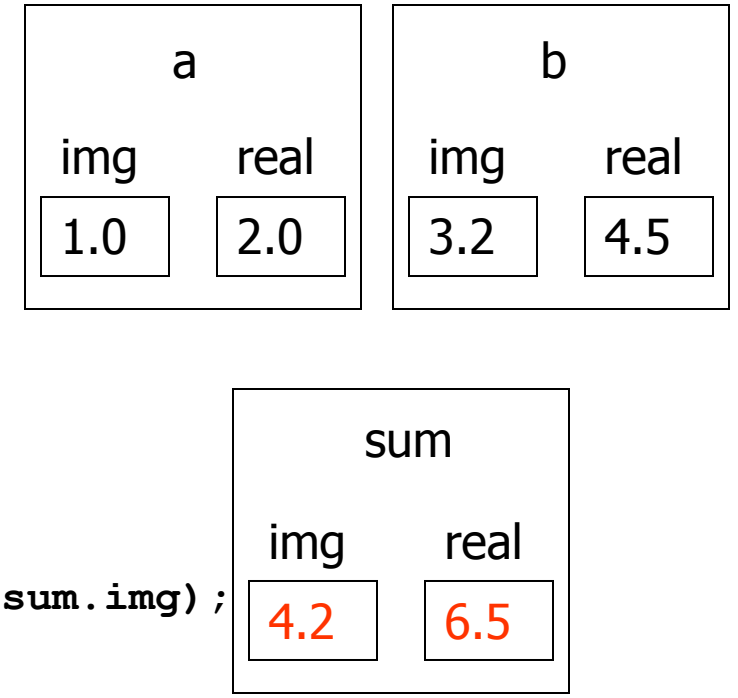






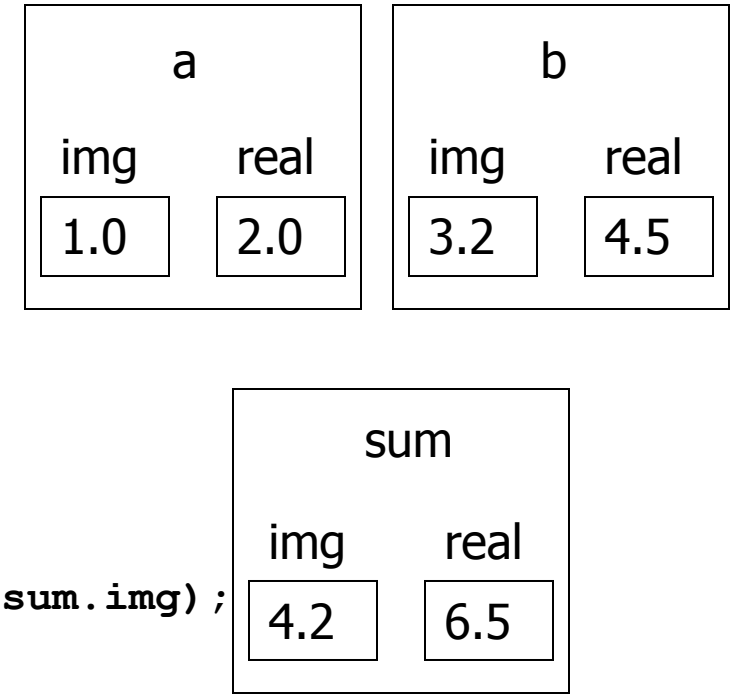
add_complex – step by step

```
int main(void)
{
    Complex a, b, sum;

    printf("…");
    scanf("%lf%lf", &(a.real), &(a.img));
    printf("…");
    scanf("%lf%lf", &(b.real), &(b.img));

    sum = add_complex(a, b);

    printf("result = %g+%gi\n", sum.real, sum.img);

    return 0;
}
```



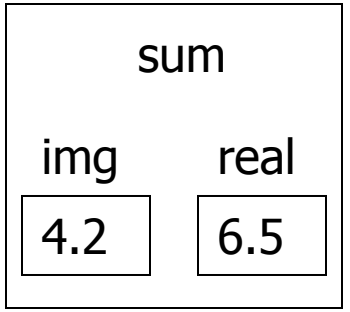
add_complex – step by step

```
int main(void)
{
    Complex a, b, sum;

    printf("…");
    scanf("%lf%lf", &(a.real), &(a.img));
    printf("…");
    scanf("%lf%lf", &(b.real), &(b.img));

    sum = add_complex(a, b);

    printf("result = %g+%gi\n", sum.real, sum.img);

    return 0;
}
```



add_complex – step by step

```
int main(void)
{
    Complex a, b, sum;

    printf("…");
    scanf("%lf%lf", &(a.real), &(a.img));
    printf("…");
    scanf("%lf%lf", &(b.real), &(b.img));

    sum = add_complex(a, b);

    printf("result = %g+%gi\n", sum.real, sum.img);

    return 0;
}
```



Exercise

- Implement a multiply_complex function:

```
Complex multiply_complex(Complex x, Complex y);
```

  - Note: if $x = a + bi$ and $y = c + di$ then:
    $z = xy = (ac - bd) + (ad + bc)i$

- Write a program that uses the above function to multiply two complex numbers given by the user

Pointers to Structs

- Pointer definition

```
Complex c, *pcomp;
pcomp = &c;
```

- To access the fields we can write:

```
(*pcomp).real
(*pcomp).img
```

- OR

```
pcomp->real
pcomp->img
```

Example – print a polynom

- Each element in the polynom is called monom

$$3x^4 + 6x^2 + 5$$

```
typedef struct monom{
  double coefficient;
  int degree;
}Monom;
```

Example – print a polynom

- Write a function that gets a **valid** polynom (pointer to array of Monoms) and the size of the polynom (number of monoms) and prints the polynom in the format :

$$3x\text{^}4+6x\text{^}2+5$$

```
void printPolynom(Monom * polynom, int size){
  int i;
  for (i=0;i<size;i++){
     if(polynom[i].coefficient != 0){
       if(i> 0) printf(" + ");
       if(polynom[i].coefficient != 1 ||
          polynom[i].degree == 0)
         printf("%g", polynom[i].coefficient);
       if(polynom[i].degree != 0)
         printf("x");
       if(polynom[i].degree > 1)
         printf("^%d", polynom[i].degree);
     }}
printf("\n");}
```

Example (struct in struct)

```
typedef struct point
{
    double x;
    double y;
} Point;

typedef struct circle
{
    Point  center;
    double radius;
} Circle
```

Example

A point is in a circle if its distance from the center is smaller than the radius.

```
int distance(point * p1, point * p2){
    return sqrt( (p1->x - p2->x) * (p1->x - p2->x) +
                 (p1->y - p2->y) * (p1->y - p2->y));
}

int is_in_circle(Point *p, Circle *c)
{
    return (distance(p, &c->center) <= c->radius);
}
```

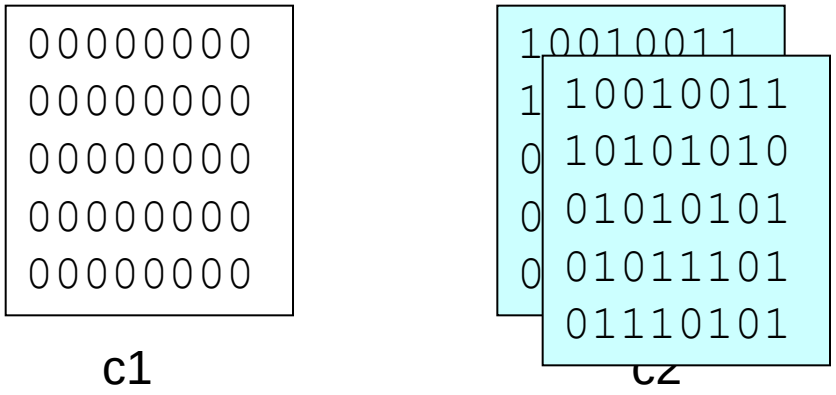
Comparison

- Structures cannot be compared using the equality operator “==”
  - They must be compared member by member
  - Usually this will be done in a separate function

```
is_equal_complex(Complex c1, Complex c2)
{
    return (c1.real == c2.real)  &&  (c1.img == c2.img);
}
```

Assignment

- Structures can be assigned (copied) using the assignment operator “=”
  - Bitwise copy – copying the content of one structure’s memory onto another
- c1 = add_complex(…);



Pointers in Structs

- When copying a struct containing a pointer only the pointer is copied (shallow copy), not what the pointer points to.
- We should take extra care when manipulating structures that contain pointers

Example (Pointers in structs)

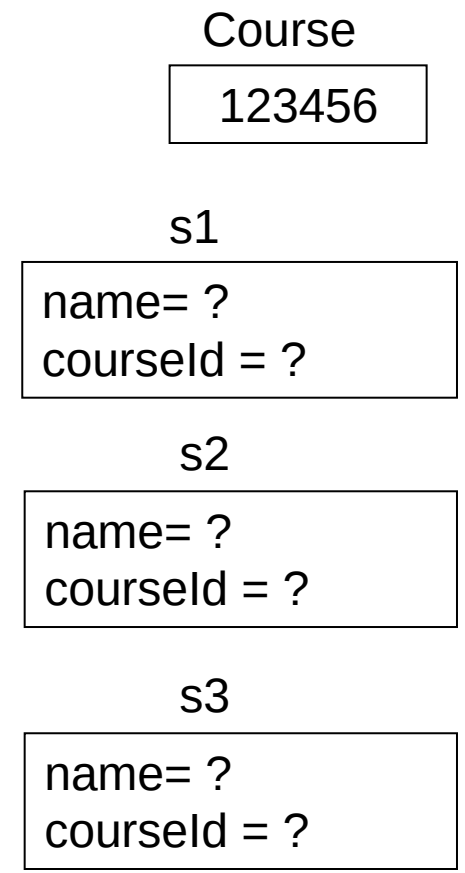
```
typedef struct student  {
  char name [MAX_LEN];
  char * courseId;
} Student;

Void initStudent(Student* s, char * name,
  char * course){
     strcpy(s->name, name);
     s->courseId = course;
}
```

```
int main()  {
  char course = "123456";

  Student s1, s2, s3;

  initStudent(&s1, "ronit", course);
  initStudent(&s2, "yossi", course);
  s3 = s2;

  strcpy(s3.name, "yaniv");
  strcpy(s3.courseId, "456789");
  return 0;
}
```

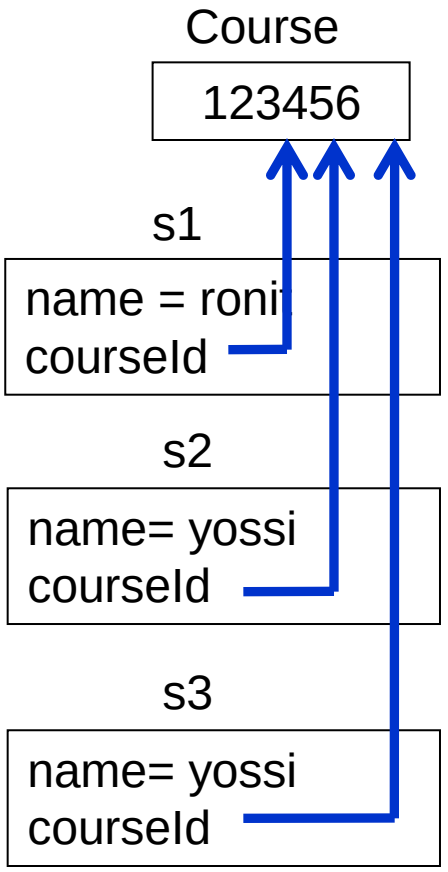
Course

123456

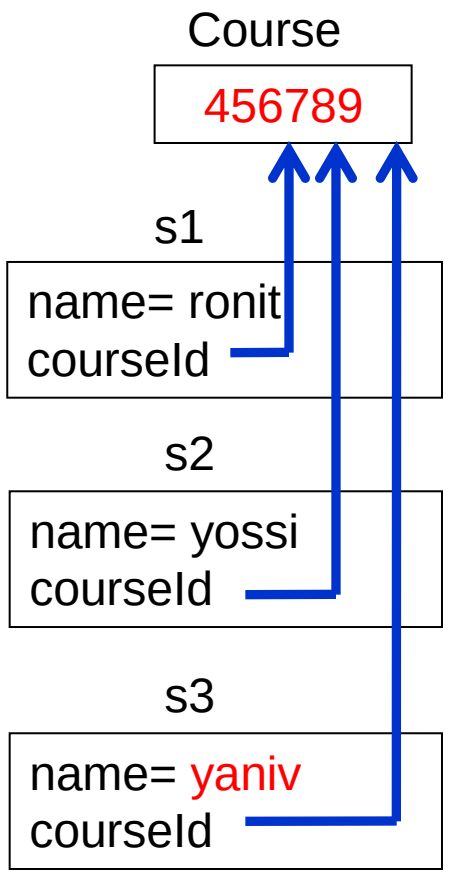
```
int main()  {
  char course = "123456";

  Student s1, s2, s3;

  initStudent(&s1,"ronit",course);
  initStudent(&s2,"yossi",course);
  s3 = s2;

  strcpy(s3.name, "yaniv");
  strcpy(s3.courseId, "456789");
  return 0;
}
```



```
int main()  {
   char course = "123456";

   Student s1, s2, s3;

   initStudent(&s1,"ronit",course);
   initStudent(&s2,"yossi",course);
   s3 = s2;

   strcpy(s3.name, "yaniv");
   strcpy(s3.courseId, "456789");
   return 0;
}
```



```
int main()  {
   char course = "123456";

   Student s1, s2, s3;

   initStudent(&s1,"ronit",course);
   initStudent(&s2,"yossi",course);
   s3 = s2;

   strcpy(s3.name, "yaniv");
   strcpy(s3.courseId, "456789");
   return 0;
}
```

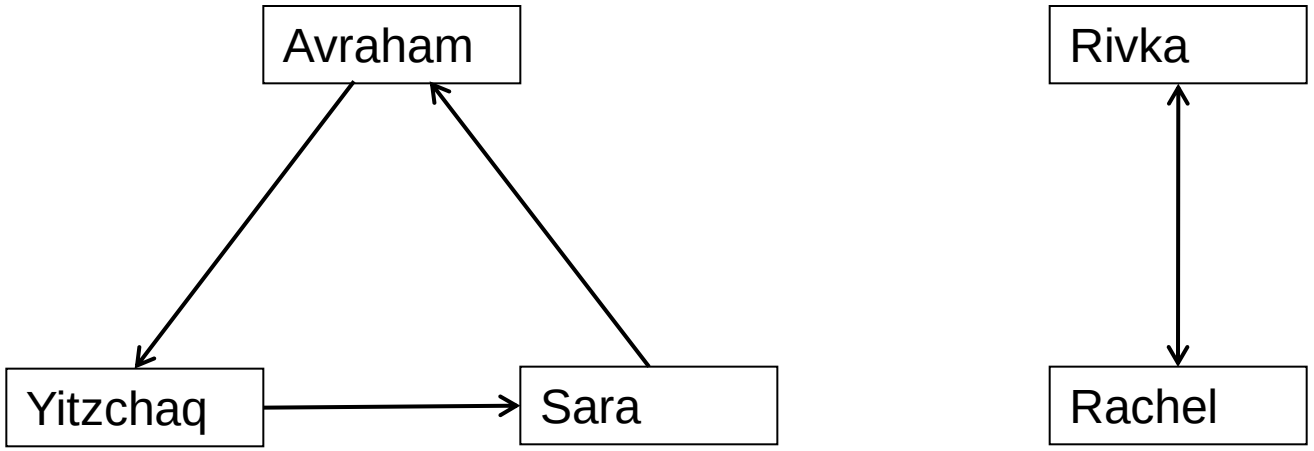


Example – Social circle (exam 2009a)

We have an array of students. Each student has a name and best friend (represented as it's index in the array).

	name: Avraham best_friend: 1	name: *Yitzchaq* best_friend: 3	name: Jacob best_friend: 0	name: Sara best_friend: 0	name: Rivka best_friend: 5	name: Rachel best_friend: 4
Index:	**0**	**1**	**2**	**3**	**4**	**5**

In our example:



Jacob

Example – Social circle – part A

```
typedef struct student {
  char name[40];
  int best_friend;
} Student;
```

- Write a function **inCircle** that checks if a student belongs to a social circle.
- The function returns TRUE/FALSE and the number of students in circle.

Solution – Social circle – part A

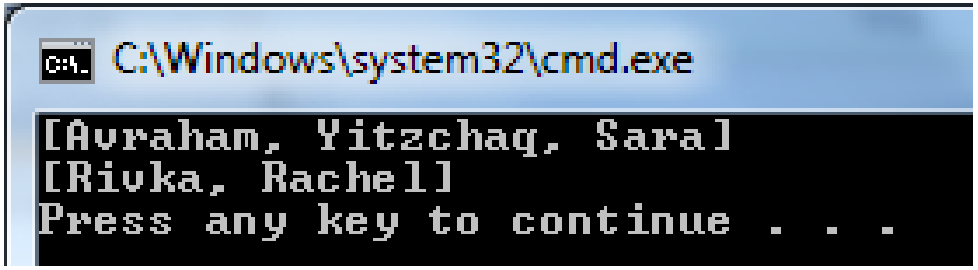
```
int inCircle(Student course[], int size, int
                              studentIndex){
  int i, curr =
     course[studentIndex].best_friend;
  for (i = 0; i < size && curr !=
     studentIndex; i++)
  {
     curr = course[curr].best_friend;
  }
  return (i != size) ? TRUE : FALSE;
}
```

Example – Social circle – part B

- Write a function **printCircle** that gets student's array, additional array ('wasPrinted'), the size of the arrays and student's index.
- If the student belongs to a social circle, the function prints it's friends and update the relevant cells in 'wasPrinted' to be TRUE.
- Otherwise, the function prints nothing.

```
void printCircle(Student course[], int
  wasPrinted[],int size, int studentIndex){
  int i;
  if (!inCircle(course, size, studentIndex))
     {printf("[]\n"); return;}
  printf("[");
  i = studentIndex;
  do{
     printf("%s", course[i].name);
     wasPrinted[i] = TRUE;
     i = course[i].best_friend;
     if (i != studentIndex)   printf(", ");
  } while (i != studentIndex);
  printf("]\n");
}
```

```
void printAllCircles(Student course[],int printed[]
        ,int size){
   int i;
   for (i = 0; i < size; i++)
       if (!printed[i])
             printCircle(course, printed, size, i);
}
int main(){
   Student students[NUM_STUDENTS] = { {"Avraham", 1},
       {"Yitzchaq", 3},{"Jacob", 0},{"Sara", 0},
       {"Rivka", 5},      {"Rachel", 4} };
   int printed[NUM_STUDENTS] = {0};
   printAllCircles(students, printed   , NUM_STUDENTS);
   return 0;
}
```



Example – char appearances

```
typedef struct appear{
    char ch;
    int n_appearances;
} appear_t;
```

- Write a function that gets a string and updates the output array **`appear`** (each cell will contain a different char and its number of appearances) . Assume that appear is big enough. The function will return the number of different chars.
-

```
int appearances (char* str, appear_t
appear[])
```

```
int appearances (char* str, appear_t appear[]){
     int i, found, n=0;
     while(*str != '\0'){
          found = FALSE;
          for (i=0;i<n;i++){
               if(appear[i].ch == *str){
                    appear[i].n_appearances++;
                    found = TRUE;
               }}
          if(!found){
               appear[n].ch = *str;
               appear[n].n_appearances = 1;
               n++;
          }
          str++;
     }
     return n;}
```

Solution - multiply_complex

```
Complex multiply_complex(Complex a, Complex b)
{
    return make_complex(a.real * b.real - a.img * b.img,
                        a.real * b.img +  a.img * b.real);
}
```