Basics of programming

Balázs Goldschmidt

Complex types: arrays

Problem

- Let's create an application that
  - Reads in 100 numbers
  - Prints out the average

```
int main() {
   int n = 100, x, i, sum = 0;
   for (i = 0; i < n; i++) {
       scanf("%d", &x);
       sum += x;
   }
   printf("%d\n", sum/n);
}
```

Problem

- Let's create an application that
  - Reads in 100 numbers
  - Prints out those less than the average
- Solution:
  - average is only known after reading in all numbers
  - numbers have to be stored
  - the store is the *array*

Array usage

declaration

left value

right value

```
int main() {
   int n = 100, x, i, sum = 0;
   int number[100];
   for (i = 0; i < n; i++) {
       scanf("%d", &x);
       sum += x;
       number[i] = x;
   }

   for (i = 0; i < n; i++) {
       if (number[i] < sum/n) {
              printf("%d\n", number[i]);
       }
   }
}
```


Formal array usage

- Declaration
  - *type name*[*size*]
    - type is any type (primitive or complex)
    - name is any identifier
    - size is any constant (compile time) integer expression
- Assignment and indexing
  - *name*[*index*] = *expr;*
    - name is name of array
    - index is any integer expression
    - expression is of the array's type

Array rules

- Size of array is constant
  - the size of an array cannot be modified
  - int x[10] stores 10 elements (or less)
- Size of array is unknown
  - the size must be stored, otherwise lost
  - there's no way of getting out the size from an array
- Indexing starts with 0
  - index of last element is *n-1*
  - indexing *n*, *n+1*, *-2*, etc. compiles but erroneous

Arrays and functions

- Parameter passing
  - Problem: let's create a function (avg)
    - input: int array
    - output: average of elements
  - Result:

```
int avg(int a[], int n) {
   int i, sum = 0;
   for (i = 0; i < n; i++) {
       sum += a[i];
   }
   return sum/n;
}
```


Arrays and functions

- Parameter passing
  - An array is always passed by *reference*
  - Consider:

```
void bar(int x[], int n) {
   int i;
   for (i = 0; i < n; i++) {
       x[i] = 2*x[i];
   }
}
...
bar(a, 10);
// a is modified!
```

Sorting arrays

- Problem:
  - write function that sorts an array
- Solution:
  - bubble sort
    - swap adjacent elements while necessary
  - selection sort
    - select maximum, put it to end, repeat
- Helper function: swap

Helper function: Swap

- Problem
  - write function that swaps two elements in an array
- Solution

```
void swap (int a[], int i, int j) {
   int tmp;
   tmp = a[i];
   a[i] = a[j];
   a[j] = tmp;
}
```


Bubble sort

- While changing, swap out-of-order adjacent elements

```
void bubble_sort(int a[], int n) {
   int i, swapped;
   do {
       swapped = 0; // false
       for (i = 0; i < n-1; i++) {
              if (a[i] > a[i+1]) {
                     swap(a, i, i+1);
                     swapped = 1; // true
              }
       }
   } while (swapped);
}
```

Selection sort

- Select maximum, put at end, start over

```
int maxIndex(int a[], int n) {
   int i,m = 0;
   for (i = 1; i < n; i++) {
       if (a[m] < a[i]) m = i;
   }
   return m;
}

void select_sort(int a[], int n) {
   int i;
   for (i = n; i > 1; i--) {
       int max = maxIndex(a, i);
       swap(a, max, i-1);
   }
}
```


Arrays and functions

- Return value
  - only with dynamic arrays
  - to be taught later…

Multidimensional arrays

- Problem: *let's create a multiplication table*
- Solution:
  - 11x11 array
    - array of arrays
    - `int t[11][11];`
  - element at indexes x,y equals x*y
    - `t[x][y] = x*y;`

Complex types: structs

Struct intro

- Problem:
  - read in 100 complex numbers
  - print out the one closest to origin
- Solution:

```
double re, im, re0, im0;
int i;
scanf("%lf %lf", &re0, &im0);
for (i = 1; i < 100; i++) {
   scanf("%lf %lf", &re, &im);
   if ((re*re+im*im) < (re0*re0+im0*im0)) {
       re0 = re;
       im0 = im;
   }
}
printf("%lf %lf\n", re0, im0);
```


Struct types

- Can we handle *re* and *im* together?
- Solution: structure

```
struct name {
       type1 name1;
       type2 name2;
       ...
};

struct name v1, v2, v3, ...;

v1.name1 = 13;
v2.name2 = v1.name2;
v3 = v1;
```

Struct example

- Definition of complex type:

```
struct complex {
   double re;
   double im;
};
```

- Definition of abs²:

```
double abs2(struct complex c) {
   return c.re*c.re+c.im*c.im;
}
```


Struct example

- Selecting complex closest to origin

```
int main() {
   complex c, c0;
   int i;
   scanf("%lf %lf", &c0.re, &c0.im);
   for (i = 1; i < 100; i++) {
       scanf("%lf %lf", &c.re, &c.im);
       if (abs2(c)<abs2(c0) {
             c0 = c;
       }
   }
   printf("%lf %lf\n", c.re, c.im);
   return 0;
}
```


Struct example2

- Problem:
  - read 100 complex numbers
  - sort them

    c1 > c2: **if** c1.re>c2.re

    or

    (c1.re==c2.re **and** c1.im > c2.im)

Struct example2

- Naïve solution without struct
  - assume double *sort* function

```
int main() {
   double re[100], im[100];
   int i;
   for (i = 0; i < 100; i++) {
       scanf("%lf %lf", &re[i], &im[i]);
   }
   sort(re, 100);
   sort(im, 100);
   for (i = 0; i < 100; i++) {
       printf("%lf %lf\n", re[i], im[i]);
   }
   return 0;
}
```


Struct example2

- Naïve cont.: *sort* for two double arrays

```
void sort(double re[], double im[], int n) {
   int i, swapped;
   do {
     swapped = 0; // false
     for (i = 0; i < n-1; i++) {
       if (re[i] > re[i+1]
           || (re[i]==re[i+1] && im[i]>im[i+1])) {
             swap(re, i, i+1);
             swap(im, i, i+1);
             swapped = 1; // true
       }
     }
   } while (swapped);
}
```


Struct example2

- Proper solution with struct
  - complex *swap* and *sort* functions:

```
void sort (struct complex c[], int n) {
   int i;
   for (i = n; i > 1; i--) {
       int max = maxIndex(a, i);
       swap(a, max, i-1);
   }
}
void swap (struct complex c[], int i, int j) {
   struct complex tmp;
   tmp = c[i];
   c[i] = c[j];
   c[j] = tmp;
}
```

Struct example2

- Proper solution with struct
  - complex *greater_than* and *maxIndex* functions:

```
int maxIndex(int a[], int n) {
   int i,m = 0;
   for (i = 1; i < n; i++) {
       if (gt(a[i],a[m])) m = i;
   }
   return m;
}
int gt(struct complex c1, struct complex c2) {
   return (c1.re > c2.re
             || (c1.re==c2.re && c1.im > c2.im);
}
```


Struct exercise

- Read in 5 complex numbers
- Print out those with greater absolute value than average
- Subproblems:
  - write function for reading in a single complex
  - write function for complex addition
  - write function for complex-double division

Type definition: *typedef*

- Motivation
  - long typenames
  - more descriptive typenames
- Solution: *typedef* keyword

```
typedef oldtype newtype;
```

```
typedef long userId;
typedef double[5][5] matrix5x5;
typedef struct complex { double re,im }
   complex;
```

```
userId id;
matrix5x5 m;
complex c1,c2,c3;
```

```
double det(matrix5x5 mat)
   { ... }
```

Enumeration

- Some problems need short set of values
  - name of months, days of week, identifiers, etc
- Type: enum
  - usually with typedef
  - directly convertible to *int* and back (starting with 0)
    - default value can be set

```
enum tday { mon=1, tue, wed, thu, fri, sat, sun=0 };
typedef enum tday day;
```

```
int book(int row, int seat, day d) { ... }
```


Enumeration example

```
enum tday { mon=1, tue, wed, thu, fri, sat, sun=0 };
typedef enum tday day;

int book(int row, int seat, day d) {
   if (d == tue) return 1;
   else return 2;
}

int foo() {
   int k;
   k = book(13, 2, fri);
   printf("%d\n", k);
}
```