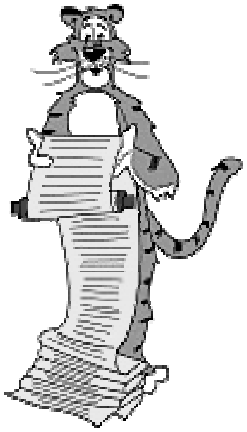
Data Structures: Arrays and Structs



Lecture P2

Why Data Structures?

Goal: deal with large amounts of data.

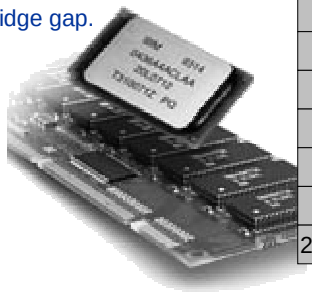
- Organize data so that it is easy to manipulate.
- Time and space efficient.

Basic computer memory abstraction.

- Indexed sequence of binary numbers.
- Address = index.

Need higher level abstractions to bridge gap.

- ARRAY
- STRUCT
- Linked list



addr	value
0	0
1	1
2	1
3	1
4	0
5	1
6	0
7	0
8	1
9	0
10	1
. . .	. . .
256GB	1

Data Types and Data Structure

Data type

- Set of possible values for variables.
- Operations on those values.
- Ex: `int, float, double, char.`

Data structure

- Collection of related values.
- Mechanism for organizing data.
- Examples.
  - built-in to language: array, struct
  - linked: linked list, binary tree
  - compound: array of structs, list of trees

Read Sedgewick 3.1.

Arrays

Fundamental data structure.

- HOMOGENEOUS collection of values (all same type).
  - vector, matrix, string of characters, spreadsheet, ...
- Store values SEQUENTIALLY in memory.
- Associate INDEX with each value.
- Concise and efficient mechanism for dealing with large collections of data values.

Memory address	107	108	109	110	111	112	113	114	115	116
Array index	0	1	2	3	4	5	6	7	8	9
Value	0	1	1	2	3	5	8	13	21	34

Arrays

Fundamental data structure.

- HOMOGENEOUS collection of values (all same type).
  - vector, matrix, string of characters, spreadsheet, ...
- Store values SEQUENTIALLY in memory.
- Associate INDEX with each value.
- Concise and efficient mechanism for dealing with large collections of data values.

Built-in to C.

- To access i^{th} element of array named `grades`, use `grades[i]`
- Caveats:
  -
  -
- Limitation: need to know size of array ahead of time.

Array Example: Manipulate Polynomials

C representation of $x^9 + 3x^5 + 7$

```
int i, a[10];
for (i = 0; i < 10; i++)
  a[i] = 0;
a[0] = 7; a[5] = 3; a[9] = 1;
```

Possible memory representation of $x^9 + 3x^5 + 7$.

- Assume array is stored in locations 107-116.

Memory address	107	108	109	110	111	112	113	114	115	116
Array index	0	1	2	3	4	5	6	7	8	9
Value	7	0	0	0	0	3	0	0	0	1

- Advantage: can get to each item quickly.
  - index carries implicit information, takes no space
- Disadvantage: uses up space for unused items.

Array Example: Strings

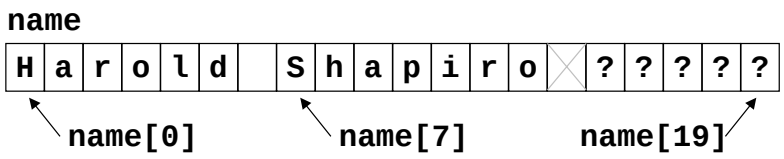
A variable of type `char` stores a character.

```
char c = 'H';
```

A STRING is an array of characters.

```
char name[20] = "Harold Shapiro";
```

- Implicitly ends with '\0' which is the same as 0.



Array Example: Yahtzee

Yahtzee is a "fast-paced, tension-filled game for one or more players" that involves rolling dice. (Milton Bradley)

- Throw five dice. If they all match you yell "YAHTZEE".
- See `www.yahtzee.com` for other rules.

```
int dice[5];
int i, match;

for (i = 0; i < 5; i++)
  dice[i] = randomInteger(6) + 1;

match = 1;
for (i = 1; i < 5; i++)
  if (dice[i] != dice[0]) match = 0;

if (match == 1)
  printf("YAHTZEE!\n");
```

Array Example: Computing a Histogram

Unix

```
% more histo.data
  68 69 67 65 60 68 67 94 59 54 21 96 95 94 55 69 93
  64 65 93 54 64 94 63 65 89 93 28 92 62 61 87 92 55
  89 51 95 85 88 94 86 93 93 84 86 93 84 92 93 92 85
  90 84 87 88 90 85 87 91 87 87 82 85 85 88 87 86 81
  90 85 81 85 84 78 90 85 79 91 92 75 78 89 76 91 91
  75 78 89 76 90 81 74 78 80 76 80 81 84 76 78 74 79
  79 84 74 77 84 78 76 82 70 77 83 76 74 70 83 76 81
  72 73 67 81 72 69 84 83 71 72 84 73 82 83 70 71 82
  69 82 82 73 80
% gcc histo.c; a.out < histo.data
   0- 9
  10-19
  20-29 **
  30-39
  40-49
  50-59 ******
  60-69 ******************
  70-79 **********************************
  80-89 ************************************************
  90-99 ***************************
```


Array Example: Computing a Histogram

Histogram = bar graph of number of occurrences of each value.

- Grades in range 0-99.
- How many in ranges 0-9, 10-19, 20-29, . . ., 90-99?

First, let's do without arrays. . . .

Computing a Histogram Without Arrays

increment appropriate bin

print right number of stars for each bin

```
#include <stdio.h>
int main(void) {
  int bin0 = 0, bin1 = 0,
      bin2 = 0, ..., bin9 = 0;
  int i, val;

  while (scanf("%d", &val) != EOF) {
    if (val < 10) bin0++;
    else if (val < 20) bin1++;
    else if (val < 30) bin2++;
    . . .
    else if (val < 90) bin8++;
    else bin9++;
  }

  printf("\n 0 -9 ");
  for (i = 0; i < bin0; i++)
    printf("*");
  . . .
  printf("\n90-99 ");
  for (i = 0; i < bin9; i++)
    printf("*");
  return 0;
}
```


Array Example: Computing a Histogram

Histogram = bar graph of number of occurrences of each value.

- Grades in range 0-99.
- How many in ranges 0-9, 10-19, 20-29, . . ., 90-99?

First, let's do without arrays.

- 50+ lines of repetitive code.
-

Now let's do with arrays: use key feature of arrays.

- Use data as index.
- See Program 3.7 in Sedgewick.

Array Example: Computing a Histogram

initialize array

calculate which bin
increment that bin

print right number of
stars for each bin

```
#include <stdio.h>
int main(void) {
  int i, j, val;
  int h[10];

  for (i = 0; i < 10; i++)
    h[i] = 0;

  while (scanf("%d", &val) != EOF) {
    bin = val / 10;
    h[bin]++;
  }

  for (j = 0; j < 10; j++) {
    printf("%2d-%2d ", j*10, j*10+9);
    for (i = 0; i < h[j]; i++)
      printf("*");
    printf("\n");
  }
  return 0;
}
```

hist.c

Shuffling

Goal: create a random permutation of the integers 0 through N-1.

Shuffling Algorithm 1.

- Start with an ordered array from 0 to N-1.
- Pick two indices at random, and exchange values.
- Repeat many times.
- Drawbacks.
  - slow for large N
  - not perfectly random

```
void randPerm(int a[], int N) {
  int i, j;
  for (i = 0; i < N; i++)
    a[i] = i;
  while(1) {
    i = randomInteger(N);
    j = randomInteger(N);
    t = a[i]; a[i] = a[j]; a[j] = t;
  }
}
```


Shuffling

Shuffling Algorithm 2.

- In i^{th} iteration:
  - choose random integer r between 0 and i
  - put value in position r into position i
  - put value i in position r
- Need random access to arbitrary element ⇒ use arrays.

Claim: after i^{th} iteration, array contains random permutation of integers 0 through i in array positions 0 through i.

```
void randomPermutation(a[], int N) {
  int i, r;
  for (i = 0; i < N; i++) {
    r = randomInteger(i + 1);
    a[i] = a[r];
    a[r] = i;
}
```


Shuffling Applications

Applications:

- Shuffling a deck of cards.
- Assigning numbers in a housing lottery.
- Hat problem.
  - N people go to a party and take off their hats
  - hats are mixed up, and each person randomly selects one hat
  - what is likelihood that no one selects their own hat?

A sundial hat.

Hat Problem

One simulation:

- Select a random permutation of N elements.
- Check to see if any value matches it index.

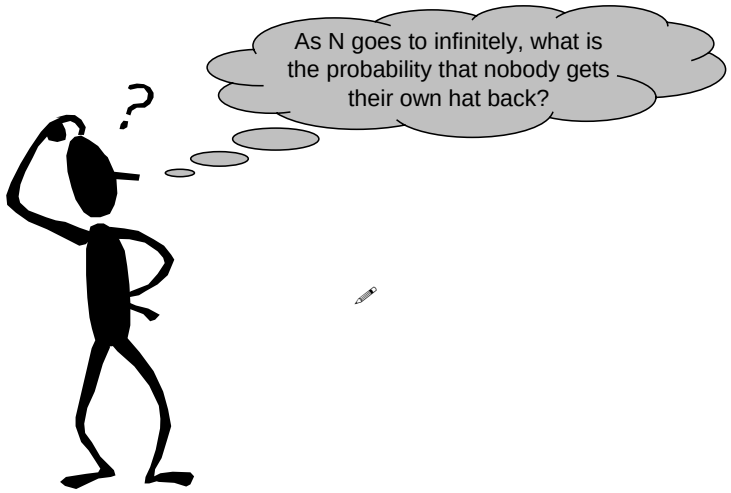
hat.c

```
#define N  27
#define TRIALS 100

int main(void) {
  int i, cnt = 0, a[N];

  for (i = 0; i < TRIALS; i++) {
    randPerm(a, N);
    for (j = 0; j < N; j++)
      if (a[j] == j) {
        cnt++;
        break;
      }
  }
  printf("fraction of failures = %f\n",
    1.0 * (TRIALS – cnt) / TRIALS);
  return 0;
}
```

`(a[j] == j)` if j gets their own hat

once you find a match, break out of loop

Hat Problem



Array Example: The Birthday Problem

People enter an empty room until a pair of people share a birthday. How long will it take on average?

- Assume birthdays are uniformly random integers between 0 and 364.

b[d] = 0 if day d not "filled"
= 1 if day d is "filled"

Mark all birthdays as not "filled".

If birthday d is "filled" return. Otherwise mark d as "filled."

bday.c

```
#define DAYS 365
int b[DAYS];

int bday(void) {
  int i;
  for (i = 0; i < DAYS; i++)
    b[i] = 0;
  for (i = 0; i < DAYS; i++) {
    d = randomInteger(DAYS);
    if (b[d] == 1) return i;
    else b[d] = 1;
  }
}
```


Array Example: The Birthday Problem

People enter an empty room until a pair of people share a birthday. How long will it take on average?

- Assume birthdays are uniformly random integers between 0 and 364.

run simulation several times

bday.c

```
#include <stdio.h>
#define DAYS   365
#define TRIALS 100
int b[DAYS];

int RandomInteger(int a, int b) { . . .}
int bday(void) { … }

int main(void) {
  int i;
  for (i = 0; i < TRIALS; i++)
    printf("%d\n", bday());
  return 0;
}
```

Structs

Fundamental data structure.

- HETEROGENEOUS collection of values (possibly different type).
  - Database records, linked list nodes (see Lecture P7).
- Store values in FIELDS.
- Associate NAME with each field.
- Use struct name and field name to access value.

Built-in to C.

- To access `grade` field of structure `x` use `x.grade`
- Basis for building "user-defined types" in C.

C Representation of C Students

struct code

```
include <stdio.h>

struct student {
   char name[20];
   int grade;
}

int main(void) {
   struct student t;
   struct student x = {"Bill Gates", 60};
   struct student y = {"Steve Jobs", 70};

   if (x.grade > y.grade) t = x;
   else t = y;
   printf("Better student %s\n", t.name);

   return 0;
}
```

struct declaration

can initialize struct fields in declaration

access structs as ordinary variables

%s for string

Typedef

User definition of type name.

- Put type descriptions in one place - makes code more portable.
- Avoid typing `struct` - makes code more readable.

```
typedef int Grade;
typedef char Name[20];

struct student {
  Name name;
  Grade grade;
}

typedef struct student Student;

. . .

Student x = {"Bill Gates", 60};
```


Rational Numbers

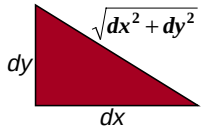
See Assignment 3.

- You will add associated operations (add, multiply, reduce) to Rational number data type.

```
typedef struct {
  int p;    /* numerator   */
  int q;    /* denominator */
} Rational;

. . .

float z;
Rational t;
. . .
z = 1.0 * t.p / t.q;
```

Geometry: Points

Define structures for points in the plane.



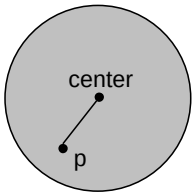
random point with x and y coordinates between –1 and 1

point data structure

```
#include <math.h>
typedef struct {
  double x;
  double y;
} Point;

double distance(Point a, Point b) {
  double dx = a.x - b.x;
  double dy = a.y - b.y;
  return sqrt(dx*dx + dy*dy);
}
Point randomPoint(void) {
  Point p;
  p.x = randomDouble(-1.0, 1.0);
  p.y = randomDouble(-1.0, 1.0);
  return p;
}
```


Geometry: Circles

Define structures for circles.



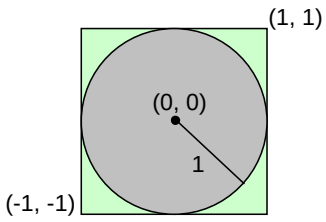
circle data structure

```
#include <math.h>

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


int inCircle(Point p, Circle c) {
  return distance(p, c.center) <= c.radius;
}
```


Geometry: Estimating Pi

Estimate pi.

- Generate N random points with x and y coordinates between -1.0 and 1.0.
- Determine fraction that lie in unit circle.
- On average pi / 4 fraction should lie in circle.
- Use 4 * fraction as (rough) estimate of pi.



pi.c

```
#define N 10000

int main(void) {
  int i, cnt = 0;
  Point p = {0.0, 0.0};
  Circle c;
  c.center = p; c.radius = 1.0;

  for (i = 0; i < N; i++) {
    p = randomPoint();
    if (inCircle(p, c)) cnt++;
  }

  printf("estimate of pi = %f ",
         4.0 * cnt / N);
  return 0;
}
```


Pass By Value, Pass By Reference

Arrays and structs are passed to functions in very DIFFERENT ways.

Pass-by-value:

- int, float, char, struct
- a COPY of value is passed to function

```
void mystery(Point a) {
  a.y = 17.0;
}

Point a = {1.0, 2.0};
mystery(a);
printf("%4.1f\n", a.y);
```

Unix

```
% a.out
 1.0
```

Pass-by-reference:

- arrays
- function has direct access to array elements

```
void mystery(double a[]) {
  a[1] = 17.0;
}

double a[] = {1.0, 2.0};
mystery(a);
printf("%4.1f\n", a[1]);
```

Unix

```
% a.out
17.0
```

Conclusions

Basic computer memory abstraction.

- Indexed sequence of binary numbers.
- Address = index.

Need higher level abstractions to bridge gap.

- Array
  - homogeneous collection of values (all same type)
  - store values sequentially in memory
  - associate index with each value
- Struct
  - heterogeneous collection of values (possibly different type)
  - Store values in fields
  - associate name with each field