Lecture P3: Arrays

Arrays

Built into C.
- Declare using [].

\[
\begin{align*}
\text{double } a0, a1, a2, a3, a4, a5, a6, a7, a8, a9; \\
\text{vs.} \\
\text{double } a[10];
\end{align*}
\]

- To access element \(i\) of array named \(a\), use \(a[i]\).
- Caveats:
  - Limitation: need to fix size of array ahead of time.

Caveats:
- Array indices start at 0 not 1.
- "Ghastly error" to access element 10 of a 10 element array.

Limitation: need to fix size of array ahead of time.

Array Example: Manipulate Polynomials

Possible memory representation (assuming array starts at 107).

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>107</th>
<th>108</th>
<th>109</th>
<th>110</th>
<th>111</th>
<th>112</th>
<th>113</th>
<th>114</th>
<th>115</th>
<th>116</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td>(c[0])</td>
<td>(c[1])</td>
<td>(c[2])</td>
<td>(c[3])</td>
<td>(c[4])</td>
<td>(c[5])</td>
<td>(c[6])</td>
<td>(c[7])</td>
<td>(c[8])</td>
<td>(c[9])</td>
</tr>
<tr>
<td>Value</td>
<td>7.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>3.8</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.2</td>
</tr>
</tbody>
</table>

Array Example: Manipulate Polynomials

Evaluating \(p(x)\) at \(x = 3.14\).

Clever, efficient alternative. (Horner's method)
Array Example: Manipulate Polynomials

\[ p(x) = c_9 x^9 + c_8 x^8 + c_7 x^7 + c_6 x^6 + c_5 x^5 + c_4 x^4 + c_3 x^3 + c_2 x^2 + c_1 x + c_0 \]

Differentiating.
\[ d(x) = p'(x). \]
\[ d(x) = 9c_9 x^8 + 8c_8 x^7 + 7c_7 x^6 + 6c_6 x^5 + 5c_5 x^4 + 4c_4 x^3 + 3c_3 x^2 + 2c_2 x + c_1 \]

```
double d[10];
int i;
for (i = 0; i < 9; i++)
    d[i] = (i + 1) * c[i + 1];
d[9] = 0.0;
```

Array Tradeoffs

**Advantage.**
- Can get to each item quickly.

**Disadvantage.**
- Consumes space for unused items.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>7.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>3.8</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.2</td>
<td></td>
</tr>
</tbody>
</table>

LFBSR Revisited

All the b variables behave the same. Why not bundle together?

```
#include <stdio.h>
#define N 100

int main(void) {
    int i, new;
    int b[10] = {0, 1, 0, 0, 0, 0, 1, 0, 1, 0};
    for (i = 0; i < N; i++) {
        new = b[3] ^ b[10]; // ^ means XOR in C
        for (j = BITS - 1; j >= 1; j--) // shift bits
            b[j] = b[j-1];
        printf("%d", new);
    }
    return 0;
}
```

```
#include <stdio.h>
#define N 100
#define BITS 11

int main(void) {
    int i, j, new;
    int b[BITS] = {0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0};
    for (i = 0; i < N; i++) {
        new = b[3] ^ b[10]; // XOR
        for (j = BITS - 1; j >= 1; j--) // shift bits
            b[j] = b[j-1];
        printf("%d", new);
    }
    return 0;
}
```
Array Example: Strings

A variable of type `char` stores a character.

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

A STRING is an array of characters.

```c
char name[20] = "Shirley Tilghman";
```

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

```c
name
```

Benford's Law

```c
#include <stdio.h>

int leadingDigit(int x)
{
    while (x >= 10)
        x /= 10;
    return x;
}

int main(void)
{
    int i, d, x, tot = 0, count[10] = {0};
    while (scanf("%d", &x) == 1) {
        d = leadingDigit(x);
        count[d]++;
        tot++;
    }
    for (i = 1; i < 10; i++)
        printf("%d: %f\n", i, 1.0 * count[i] / tot);
    return 0;
}
```

Newcomb (1881).
- Tables of logarithms.

Benford (1938).
- River area. Population.
- Newspaper. Specific heat.
- Pressure. Atomic weight.
- Drainage. Reader’s Digest.
- Baseball. Black body.
- Death rates. Addresses.

Scale invariant!

- Distribution of distributions.

Unix

```bash
% more princeton-files.txt
96796
4171208
5830
34343656
...

% gcc benford.c
% a.out < princeton-files.txt
1: 0.308
2: 0.193
3: 0.130
4: 0.099
5: 0.075
6: 0.060
7: 0.052
8: 0.044
9: 0.004
```
The First-Digit Phenomenon


\[ P_d = \log_{10} \left( \frac{1}{d} \right) \]

Predicted frequencies:

- 30.1
- 17.6
- 12.5
- 9.7
- 7.9
- 6.7
- 5.8
- 5.1
- 4.6

Sorting

Goal: given N items, rearrange them in increasing order.

Applications:
- Sort a list of names.
- Find duplicates in a mailing list.
- Find the median.
- Identify statistical outliers.

Insertion Sort

Insertion sort.
- In ith iteration:
  - read ith value
  - repeatedly swap ith value with the one to its left if it is smaller

Property: after ith iteration, array positions 0 through i contain original elements 0 through i in increasing order.

```c
#include <stdio.h>
#define N 10

int main(void) {
    int i, j;
    double swap, x[N];
    for (i = 0; i < N; i++)
        scanf("%lf", &x[i]);
    for (i = 0; i < N; i++) {
        for (j = i; j > 0; j--)
            if (x[j-1] > x[j]) {
                swap = x[j]; x[j] = x[j-1]; x[j-1] = swap;
            }
    }
    for (i = 0; i < N; i++)
        printf("%f\n", x[i]);
    return 0;
}
```

insertion-sort.c
Array Function Example: Shuffling

Goal: shuffle n-element array.

- In ith iteration:
  - choose random integer r between 0 and i
  - swap values in positions r and i
- Need random access to arbitrary element ⇒ use arrays.

Property: after ith iteration, array positions 0 through i contain random permutation of elements 0 through i.

```c
void shuffle(double a[], int n) {
    int i, r;
    double swap;
    for (i = 0; i < n; i++) {
        r = randomInteger(i + 1);
        swap = a[r];
        a[r] = a[i];
        a[i] = swap;
    }
}
```

Bicycle Problem

Bicycle problem.

- N kids go to a party and dump bicycle in a pile.
- Kids are blindfolded, and each one selects a bike at random.
- What is likelihood that at least one gets their own bike?

```c
#define N 10000
#define TRIALS 1000
int main(void) {
    int i, j, count = 0, a[N];
    for (i = 0; i < TRIALS; i++) {
        randomPermutation(a, N);
        for (j = 0; j < N; j++)
            if (a[j] == j) {
                count++;
                break;
            }
    }
    printf("successes ratio = %f\n", 1.0 * count / TRIALS);
    return 0;
}
```

A Helper Function

Create a random permutation of integers 0 through n-1.

- Fill up array with elements 0 through n-1.
- Shuffle the array.

```c
void randomPermutation(int a[], int n) {
    int i;
    for (i = 0; i < n; i++)
        a[i] = i;
    shuffle_int(a, n);
}
```
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 uniform random integers between 0 and 364.

```
#include <stdio.h>
#include <stdlib.h>
#define DAYS 365
#define TRIALS 100

int randomInteger(int n) {
    ...}

int bday(void) {
    int i, d, b[DAYS];
    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;
        }
    }
    return 0;
}

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

bday.c

run simulation several times