

Beginning C Programming for Engineers

R. Lindsay Todd

Rensselaer Polytechnic Institute

Lecture 5: Arrays

Outline

- 1 Subscripted Variable
- 2 Arrays
 - Arrays
 - Reverse, fixed
 - Reverse, varying
- 3 Array operations
 - Arrays vs. variables
 - Limitations of array operations
 - Arrays and Functions
 - Fixed-length Array Parameters
 - Example: "Reverse" as a function
 - Varying-length Array Parameters
 - Adding vectors
- 4 Multidimensional arrays
 - Matrix transpose
 - Tic Tac Toe — Examples
- 5 Character Strings
 - String I/O

Subscripted Variables

- Mathematical expressions use *subscripted variables*, e.g., a_1, a_2, \dots, a_i , to indicate a set of indexed variables.
- We can calculate indices, rather than rely on constants:

$$a = \sum_{i=0}^n V_i$$

- We can even use expressions as indices:

$$o = \sum_{k=0}^n V_{2k+1}$$

Need for Arrays

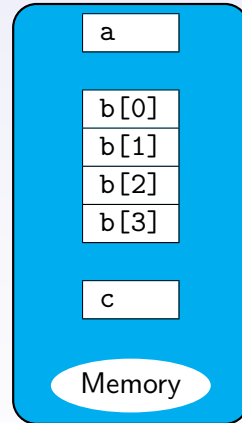
- Suppose we wanted to write a card game. We have to keep track of 52 cards in the deck:
 - ▶ What order are the cards?
 - ▶ Which cards have already been seen?
- One "solution"—Represent card_{*i*} for $0 \leq i \leq 51$:

```
int card00, card01, card02, card03, card04, card05, card06;  
int card07, card08, card09, card10, card11, card12, card13;  
int card14, card15, card16, card17, card18, card19, card20;  
int card21, card22, card23, card24, card25, card26, card27;  
int card28, card29, card30, card31, card32, card33, card34;  
int card35, card36, card37, card38, card39, card40, card41;  
int card42, card43, card44, card45, card46, card47, card48;  
int card49, card50, card51;
```

- Now assign each card_{*i*} a different random number between 1 and 52.

Arrays

- C's *arrays* are analogous to subscripted variables.
- Arrays are declared using the normal type declarations, with the size specified in brackets.
- Elements are indexed using a subscript expression in brackets.
- Subscripts range from 0 to one less than the bound.



```
int a;
int b[4];
int c;
```

C

Using Arrays

- Now we can approach the card problem.
- Can represent card_{*i*}, for $0 \leq i \leq 51$, and call a function:

```
int card[52];
deal(card);
```

Can use loops in `deal`:

```
void deal(int card[52]) {
    int i, j, tmp;
    for (i = 0; i < 52; ++i) {
        card[i] = i;
    }
    for (i = 0; i < 52; ++i) {
        j = rand() % 52;
        tmp = card[j];
        card[j] = card[i];
        card[i] = tmp;
    }
}
```

C

Example: Reverse Number List?

- We wish to read in numbers from the keyboard, then print them in reverse order.
- Clearly, we need to know all the numbers before we can print them! So they all need to be stored.
- We store in an array, which implies we set a limit on the number of entries we can read.

C

Code: Reverse Number List

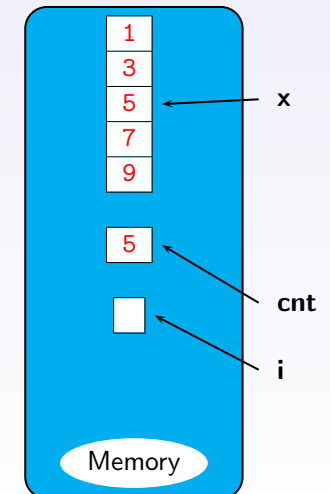
Program: `reverse1.c`

```
#include <stdio.h>

int main()
{
    int x[5], cnt, i;

    for (cnt = 0; cnt < 5; ++cnt) {
        scanf("%d", &x[cnt]);
    }
    for (i = 4; i >= 0; i--) {
        printf("%d\n", x[i]);
    }
    return 0;
}
```

```
1
3
5
7
9
9
7
5
3
1
```



C

Example: Varying Length Lists

- We wish to read in numbers from the keyboard, then print them in reverse order. We keep reading until we indicate there is no more input.
- Still need an array, but now we also need to keep track of how many entries in the array are occupied.

Code: Reverse Varying Number List

Program: reverse2a.c

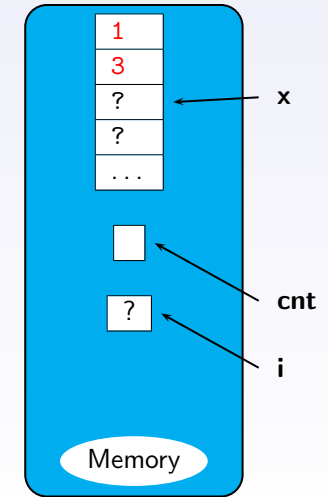
```
#include <stdio.h>

#define MAXSIZE 500

int main()
{
    int x[MAXSIZE], cnt, i, v;

    for (cnt = 0; cnt < MAXSIZE; ++cnt) {
        scanf("%d", &v);
        if (v < 0) {
            break;
        }
        x[cnt] = v;
    }
    for (i = cnt-1; i >= 0; i--) {
        printf("%d\n", x[i]);
    }
    return 0;
}
```

```
1
3
-1
3
1
```



Arrays vs. Variables

Arrays are treated differently than “simple” variables:

- Elements of arrays must be *deferenced*, e.g., have a subscript, before they can be accessed.
- Simple variables must not be dereferenced with a subscript.

```
/* M an array */
float M[1];
M[0] = 1.0; /* Valid */
M = 2.0; /* Not valid */
```

```
/* M is simple */
float M;
M[0] = 1.0; /* Not valid */
M = 2.0; /* Valid */
```

Limitations of array operations

- There are no built-in operations on arrays, such as assignment, math, or relational operators.
- To operate on an entire array, use functions.
- Functions cannot return arrays.
- Arrays passed as arguments are passed *by reference*.

```
float x[5], y[5], z[5];
/* Omitted other stuff... */
if (x != y) {
    /* No testing for equality of arrays! */
    z = x + y;
    /* No assignment, addition of arrays! */
}
```

```
float x[5], y[5], z[5];
/* Omitted other stuff... */
if (!same(x, y)) {
    addvec(x, y, z);
}
```

Passing Arrays to Functions

Program: dotprod2.c

```
#include <stdio.h>

float dotprod2(float x[2], float y[2]);

int main() {
    float a[2], b[2];
    float d;

    printf("Ax, Ay: ");
    scanf("%f, %f", &a[0], &a[1]);

    printf("Bx, By: ");
    scanf("%f, %f", &b[0], &b[1]);

    d = dotprod2(a, b);
    printf("Dot product: %f\n", d);
    return 0;
}

float dotprod2(float x[2], float y[2])
{
    return x[0]*y[0] + x[1]*y[1];
}
```

Program: dotprod3.c

```
#include <stdio.h>

float dotprod3(float x[3], float y[3]);

int main() {
    float a[3], b[3];
    float d;

    printf("Ax, Ay, Az: ");
    scanf("%f, %f, %f",
          &a[0], &a[1], &a[2]);

    printf("Bx, By, Bz: ");
    scanf("%f, %f, %f",
          &b[0], &b[1], &b[2]);

    d = dotprod3(a, b);
    printf("Dot product: %f\n", d);
    return 0;
}

float dotprod3(float x[3], float y[3])
{
    return x[0]*y[0] + x[1]*y[1]
        + x[2]*y[2];
}
```

Code: "Reverse" as a function

Program: reversefa.c

```
#include <stdio.h>

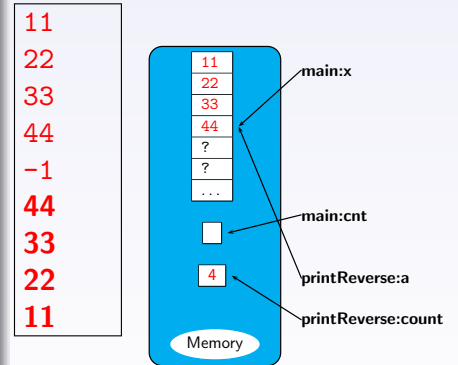
#define MAXSIZE 500

void printReverse(int a[MAXSIZE], int count);

int main()
{
    int x[MAXSIZE], cnt, v;

    for (cnt = 0; cnt < MAXSIZE; ++cnt) {
        scanf("%d", &v);
        if (v < 1) {
            break;
        }
        x[cnt] = v;
    }
    printReverse(x, cnt);
    return 0;
}

void printReverse(int a[MAXSIZE], int count)
{
    while (count > 0) {
        printf("%d\n", a[--count]);
    }
}
```



Varying-length Array Parameters

Program: dotprod2n.c

```
#include <stdio.h>

float dotprod(int n, float x[], float y[]);

int main() {
    float a[2], b[2];
    float d;

    printf("Ax, Ay: ");
    scanf("%f, %f", &a[0], &a[1]);

    printf("Bx, By: ");
    scanf("%f, %f", &b[0], &b[1]);

    d = dotprod(2, a, b);
    printf("Dot product: %f\n", d);
    return 0;
}
```

Program: dotprod.c

```
float dotprod(int n, float x[], float y[])
{
    int i;
    float sum = 0;
    for (i = 0; i < n; ++i) {
        sum += x[i]*y[i];
    }
    return sum;
}
```

- For function parameters, the array bounds may be omitted.
- A separate parameter may be needed to indicate how many elements were set.

Example: Adding vectors

Program: addvec.c

```
#include <stdio.h>

void addvec(int n,
            float a[], float b[],
            float c[]);

int main()
{
    float a[2], b[2], z[2];
    printf("Enter a0, a1: ");
    scanf("%g, %g", &a[0], &a[1]);
    printf("Enter b0, b1: ");
    scanf("%g, %g", &b[0], &b[1]);
    addvec(2, a, b, z);
    printf("Sum: %g, %g\n", z[0], z[1]);
    return 0;
}
```

Program: f-addvec.c

```
void addvec(int n,
            float a[], float b[],
            float c[])
{
    int i;
    for (i = 0; i < n; ++i) {
        c[i] = a[i] + b[i];
    }
}
```

Results

```
Enter a0, a1: 1.4, 1.3
Enter b0, b1: 2.2, 2.5
Sum: 3.6, 3.8
```

Multidimensional Arrays

- Multidimensional arrays are declared with more than one set of bracketed bounds, and referenced using more than one subscript:

```
float x[2][2];
x[0][0] = 1.0;
```

- In declaring parameters, only the first index may omit the bounds:

```
void Sort(float M[][2], int n);
```

C

Example — Matrix transpose

Program: transpose.c

```
#include <stdio.h>

void Transpose(float A[2][2]);

int main()
{
    float x[2][2];
    printf("x[0][0] : x[0][1]? ");
    scanf("%f : %f", &x[0][0], &x[0][1]);
    printf("x[1][0] : x[1][1]? ");
    scanf("%f : %f", &x[1][0], &x[1][1]);
    Transpose(x);
    printf("%f : %f\n%f : %f\n",
           x[0][0], x[0][1],
           x[1][0], x[1][1]);
    return 0;
}

void Transpose(float A[2][2])
{
    float swap;
    swap = A[0][1];
    A[0][1] = A[1][0];
    A[1][0] = swap;
}
```

Results

```
x[0][0] : x[0][1]? 1 : 2
x[1][0] : x[1][1]? 3 : 4
1.000000 : 3.000000
2.000000 : 4.000000
```

C

Tic Tac Toe: TryWinLine

Program: ttt-TryWinLine.c

```
int TryWinLine(int board[3][3], int us,
              int r1, int c1,
              int r2, int c2,
              int r3, int c3)
{
    if (board[r1][c1] == us
        && board[r2][c2] == us
        && board[r3][c3] == 0) {
        board[r3][c3] = us;
        return 1;
    }
    if (board[r1][c1] == us
        && board[r3][c3] == us
        && board[r2][c2] == 0) {
        board[r2][c2] = us;
        return 1;
    }
    if (board[r2][c2] == us
        && board[r3][c3] == us
        && board[r1][c1] == 0) {
        board[r1][c1] = us;
        return 1;
    }
    return 0;
}
```

- This function tries to make a winning move. It returns 1 if it did, 0 otherwise.
- The **board** positions are 0 (blank), 1 (X) and 2 (O). The piece we are playing with is in **us**, either 1 or 2.
- The points to check are at the coordinates (**r1,c1**), (**r2,c2**), and (**r3,c3**). For example, (0,2), (1,1), (2,0) will check one of the diagonals.

C

Tic Tac Toe: TryWinBoard

Program: ttt-TryWinBoard.c

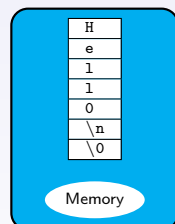
```
int TryWinBoard(int board[3][3], int us)
{
    int i;
    for (i = 0; i < 3; ++i) {
        if (TryWinLine(board, us,
                       i,0, i,1, i,2)) {
            return 1;
        }
        if (TryWinLine(board, us,
                       0,i, 1,i, 2,i)) {
            return 1;
        }
    }
    if (TryWinLine(board, us,
                   0,0, 1,1, 2,2)) {
        return 1;
    }
    if (TryWinLine(board, us,
                   0,2, 1,1, 2,0)) {
        return 1;
    }
    return 0;
}
```

- This function tries to make a winning move on the board. It returns 1 if it did, 0 otherwise.
- First this goes through to check rows and columns.
- Then the two diagonals are checked separately.

C

Character Strings

- The *ASCII* code maps printable characters to integers which can be stored in the `char` type.
- Character constants are integers that may be compared or assigned to characters.



"Hello\n"

```
char x;
x = 'a';
```

- *Strings* are arrays whose elements are type `char`.

c

String I/O

- The dangerous `gets` function reads a line as a string. It returns the string, or `NULL` on end-of-file or error. Make sure your character array is long enough to hold the data!!!!
- The `puts` function writes a string as a line of text (adding the newline). Strings can also be printed with `printf`.

Program: `chars.c`

```
#include <stdio.h>

int
main()
{
    char    s[100];

    if (gets(s) == NULL) {
        return 0;
    }
    printf("-(%c)-->%s<--\n", s[0], s);
    puts(s);
    puts("Good bye, 100%.");
    return 0;
}
```

Results

```
sample text
-(s)-->sample text<--
sample text
Good bye, 100%.
```

c

String Functions

The `string.h` header file makes functions that work with strings available.

`strlen(a)` Number of characters in `a`.

`strcmp(a,b)` Compare `a` and `b`, returning 0 if they are equal, something less than 0 if `a` is "less than" `b`, and something greater than 0 if `a` is "greater than" `b`.

`strcpy(a,b)` Copies contents of `b` into `a`.

c

Example

Program: `str.c`

```
#include <stdio.h>
#include <string.h>

int
main()
{
    char    a[100], b[100];
    int    bLen;
    a[0] = '\0';

    while (gets(b) != NULL) {
        if (!strcmp(a, b)) {
            printf("same ");
        }
        bLen = strlen(b);
        printf("%d\n", bLen);
        strcpy(a, b);
    }
    return 0;
}
```

Results

```
abcdef
6
hijk
4
hijk
same 4
```

c

reverse1.c

```
#include <stdio.h>

int main()
{
    int x[5], cnt, i;

    for (cnt = 0; cnt < 5; ++cnt) {
        scanf("%d", &x[cnt]);
    }
    for (i = 4; i >= 0; i--) {
        printf("%d\n", x[i]);
    }
    return 0;
}
```

reverse2a.c

```
#include <stdio.h>

#define MAXSIZE 500

int main()
{
    int x[MAXSIZE], cnt, i, v;

    for (cnt = 0; cnt < MAXSIZE; ++cnt) {
        scanf("%d", &v);
        if (v < 0) {
            break;
        }
        x[cnt] = v;
    }
    for (i = cnt-1; i >= 0; i--) {
        printf("%d\n", x[i]);
    }
    return 0;
}
```

dotprod2.c

```
#include <stdio.h>

float dotprod2(float x[2], float y[2]);

int main() {
    float a[2], b[2];
    float d;

    printf("Ax, Ay: ");
    scanf("%f, %f", &a[0], &a[1]);

    printf("Bx, By: ");
    scanf("%f, %f", &b[0], &b[1]);

    d = dotprod2(a, b);
    printf("Dot product: %f\n", d);
    return 0;
}

float dotprod2(float x[2], float y[2])
{
    return x[0]*y[0] + x[1]*y[1];
}
```

dotprod3.c

```
#include <stdio.h>

float dotprod3(float x[3], float y[3]);

int main() {
    float a[3], b[3];
    float d;

    printf("Ax, Ay, Az: ");
    scanf("%f, %f, %f",
          &a[0], &a[1], &a[2]);

    printf("Bx, By, Bz: ");
    scanf("%f, %f, %f",
          &b[0], &b[1], &b[2]);

    d = dotprod3(a, b);
    printf("Dot product: %f\n", d);
    return 0;
}

float dotprod3(float x[3], float y[3])
{
    return x[0]*y[0] + x[1]*y[1]
           + x[2]*y[2];
}
```

reversefa.c

```
#include <stdio.h>

#define MAXSIZE 500

void printReverse(int a[MAXSIZE], int count);

int main()
{
    int x[MAXSIZE], cnt, v;

    for (cnt = 0; cnt < MAXSIZE; ++cnt) {
        scanf("%d", &v);
        if (v < 1) {
            break;
        }
        x[cnt] = v;
    }
    printReverse(x, cnt);
    return 0;
}

void printReverse(int a[MAXSIZE], int count)
{
    while (count > 0) {
        printf("%d\n", a[--count]);
    }
}
```

dotprod2n.c

```
#include <stdio.h>

float
dotprod(int n, float x[], float y[]);

int main() {
    float a[2], b[2];
    float d;

    printf("Ax, Ay: ");
    scanf("%f, %f", &a[0], &a[1]);

    printf("Bx, By: ");
    scanf("%f, %f", &b[0], &b[1]);

    d = dotprod(2, a, b);
    printf("Dot product: %f\n", d);
    return 0;
}
```

dotprod3n.c

```
#include <stdio.h>

float
dotprod(int n, float x[], float y[]);

int main() {
    float a[3], b[3];
    float d;

    printf("Ax, Ay, Az: ");
    scanf("%f, %f, %f",
          &a[0], &a[1], &a[2]);

    printf("Bx, By, Bz: ");
    scanf("%f, %f, %f",
          &b[0], &b[1], &b[2]);

    d = dotprod(3, a, b);
    printf("Dot product: %f\n", d);
    return 0;
}
```

dotprod.c

```
float dotprod(int n, float x[], float y[])
{
    int i;
    float sum = 0;
    for (i = 0; i < n; ++i) {
        sum += x[i]*y[i];
    }
    return sum;
}
```

addvec.c

```
#include <stdio.h>

void addvec(int n,
            float a[], float b[],
            float c[]);

int main()
{
    float a[2], b[2], z[2];
    printf("Enter a0, a1: ");
    scanf("%g, %g", &a[0], &a[1]);
    printf("Enter b0, b1: ");
    scanf("%g, %g", &b[0], &b[1]);
    addvec(2, a, b, z);
    printf("Sum: %g, %g\n", z[0], z[1]);
    return 0;
}
```

f-addvec.c

```
void addvec(int n,  
            float a[], float b[],  
            float c[])  
{  
    int i;  
    for (i = 0; i < n; ++i) {  
        c[i] = a[i] + b[i];  
    }  
}
```

transpose.c

```
#include <stdio.h>

void Transpose(float A[2][2]);

int main()
{
    float x[2][2];
    printf("x[0][0] : x[0][1]? ");
    scanf("%f : %f", &x[0][0], &x[0][1]);
    printf("x[1][0] : x[1][1]? ");
    scanf("%f : %f", &x[1][0], &x[1][1]);
    Transpose(x);
    printf("%f : %f\n%f : %f\n",
           x[0][0], x[0][1],
           x[1][0], x[1][1]);
    return 0;
}

void Transpose(float A[2][2])
{
    float swap;
    swap = A[0][1];
    A[0][1] = A[1][0];
    A[1][0] = swap;
}
```

ttt-TryWinLine.c

```
int TryWinLine(int board[3][3], int us,
               int r1, int c1,
               int r2, int c2,
               int r3, int c3)
{
    if (board[r1][c1] == us
        && board[r2][c2] == us
        && board[r3][c3] == 0) {
        board[r3][c3] = us;
        return 1;
    }
    if (board[r1][c1] == us
        && board[r3][c3] == us
        && board[r2][c2] == 0) {
        board[r2][c2] = us;
        return 1;
    }
    if (board[r2][c2] == us
        && board[r3][c3] == us
        && board[r1][c1] == 0) {
        board[r1][c1] = us;
        return 1;
    }
    return 0;
}
```

ttt-TryWinBoard.c

```
int TryWinBoard(int board[3][3], int us)
{
    int i;

    for (i = 0; i < 3; ++i) {
        if (TryWinLine(board, us,
                       i,0, i,1, i,2)) {
            return 1;
        }
        if (TryWinLine(board, us,
                       0,i, 1,i, 2,i)) {
            return 1;
        }
    }
    if (TryWinLine(board, us,
                   0,0, 1,1, 2,2)) {
        return 1;
    }
    if (TryWinLine(board, us,
                   0,2, 1,1, 2,0)) {
        return 1;
    }
    return 0;
}
```

chars.c

```
#include <stdio.h>

int
main()
{
    char          s[100];

    if (gets(s) == NULL) {
        return 0;
    }
    printf("--(%c)-->%s<--\n", s[0], s);
    puts(s);
    puts("Good bye, 100%.");
    return 0;
}
```

str.c

```
#include <stdio.h>
#include <string.h>

int
main()
{
    char          a[100], b[100];
    int           bLen;
    a[0] = '\0';

    while (gets(b) != NULL) {
        if (!strcmp(a, b)) {
            printf("same ");
        }
        bLen = strlen(b);
        printf("%d\n", bLen);
        strcpy(a, b);
    }
    return 0;
}
```