Programming in C

Based on the Original Slides from
Politehnica International- Computer Engineering
Lecture Slides
Outline

• Arrays
  – The concept of array
  – Defining arrays
  – Initializing arrays
  – Character arrays
  – Multidimensional arrays
  – Variable length arrays
The concept of array

- Array: a set of ordered data items
- You can define a variable called x, which represents not a single value, but an entire set of values.
- Each element of the set can then be referenced by means of a number called an index number or subscript.
- Mathematics: a subscripted variable, $x_i$, refers to the $i$th element $x$ in a set
- C programming: the equivalent notation is $x[i]$
Declaring an array

• Declaring an array variable:
  – Declaring the **type of elements** that will be contained in the array—such as int, float, char, etc.
  – Declaring the **maximum number of elements** that will be stored inside the array.
    • The C compiler needs this information to determine how much memory space to reserve for the array.)
    • This must be a **constant integer value**

• The range for valid index values in C:
  – First element is at index 0
  – Last element is at index \([\text{size}-1]\)
  – It is the task of the programmer to make sure that array elements are referred by indexes that are in the valid range! The compiler cannot verify this, and it comes to severe runtime errors!
Arrays - Example

Example:
```c
int values[10];
```
Declares an array of 10 elements of type `int`

Using Symbolic Constants for array size:
```c
#define N 10
...
int values[N];
```

Valid indexes:
```c
values[0]=5;
values[9]=7;
```
Invalid indexes:
```c
values[10]=3;
values[-1]=6;
```

In memory: elements of an array are stored at consecutive locations
#include <stdio.h>
#define N 6
int main (void) {
    int values[N];
    int index;
    for (index = 0; index < N; ++index) {
        printf("Enter value of element #%i \n", index);
        scanf("%i", &values[index]);
    }
    for (index = 0; index < N; ++index) {
        printf("values[%i] = %i\n", index, values[index]);
    }
    return 0;
}
What goes wrong if an index goes out of range?

```c
#include <stdio.h>
#define NA   4
#define NB   7
int main (void) {
    int b[NB], a[NA];
    int index;
    for (index = 0; index < NB; ++index)
        b[index] = 10 + index;
    for (index = 0; index < NA+2; ++index)
        a[index] = index;

    for (index = 0; index < NA+2; ++index)
        printf("a[%i] = %i ", index, a[index]);
    printf("\n");
    for (index = 0; index < NB; ++index)
        printf("b[%i] = %i ", index, b[index]);
    printf("\n");
    return 0;
}
```
Exercise: Array of counters

- Suppose you took a telephone survey to discover how people felt about a particular television show and you asked each respondent to rate the show on a scale from 1 to 10, inclusive. After interviewing 5,000 people, you accumulated a list of 5,000 numbers. Now, you want to analyze the results.
- One of the first pieces of data you want to gather is a table showing the distribution of the ratings: you want to know how many people rated the show a 1, how many rated it a 2, and so on up to 10.
- You want to develop a program to count the number of responses for each rating.

```
ratingCounters
0 1 2 3 4 5 6 7 8 9 10
```
Exercise: Array of counters

#include <stdio.h>
int main (void) {
    int ratingCounters[11], i, response;
    for ( i = 1; i <= 10; ++i )
        ratingCounters[i] = 0;
    printf ("Enter your responses\n");
    for ( i = 1; i <= 20; ++i ) {
        scanf ("%i", &response);
        if ( response < 1 || response > 10 )
            printf ("Bad response: %i\n", response);
        else
            ++ratingCounters[response];
    }
    printf ("\n\nRating Number of Responses\n");
    printf ("------ -------------------\n");
    for ( i = 1; i <= 10; ++i )
        printf ("%4i%14i\n", i, ratingCounters[i]);
    return 0;
}
Exercise: Fibonacci numbers

// Program to generate the first 15 Fibonacci numbers
#include <stdio.h>
int main (void)
{
   int Fibonacci[15], i;
   Fibonacci[0] = 0; // by definition
   Fibonacci[1] = 1; // ditto
   for ( i = 2; i < 15; ++i )
      Fibonacci[i] = Fibonacci[i-2] + Fibonacci[i-1];
   for ( i = 0; i < 15; ++i )
      printf ("%i\n", Fibonacci[i]);
   return 0;
}
Exercise: Prime numbers

- An improved method for generating prime numbers involves the notion that a number \( p \) is prime if it is not evenly divisible by any other prime number.
- Another improvement: a number \( p \) is prime if there is no prime number smaller than its square root, so that it is evenly divisible by it.

Is \( p \) the next prime number here?

If you can find a \( \text{primes}[i] < \sqrt{p} \) that divides evenly \( p \), than \( p \) is not prime.
Include <stdio.h>
#include <stdbool.h>

Modified program to generate prime numbers
int main (void) {
    int p, i, primes[50], primeIndex = 2;
    bool isPrime;
    primes[0] = 2;
    primes[1] = 3;
    for ( p = 5; p <= 50; p = p + 2 ) {
        isPrime = true;
        for ( i = 1; isPrime && p / primes[i] >= primes[i]; ++i )
            if ( p % primes[i] == 0 )
                isPrime = false;
        if ( isPrime == true ) {
            primes[primeIndex] = p;
            ++primeIndex;
        }
    }
    for ( i = 0; i < primeIndex; ++i )
        printf ("%i ", primes[i]);
    printf ("\n");
    return 0;
}

Exercise: Prime numbers
Initializing arrays

- int counters[5] = { 0, 0, 0, 0, 0 };
- char letters[5] = { 'a', 'b', 'c', 'd', 'e' };
- float sample_data[500] = { 100.0, 300.0, 500.5 };
- The C language allows you to define an array without specifying the number of elements. If this is done, the size of the array is determined automatically based on the number of initialization elements: int counters[] = { 0, 0, 0, 0, 0 };
#include <stdio.h>
int main (void)
{
  char word[] = { 'H', 'e', 'l', 'l', 'o', '!' };  
  int i;
  for ( i = 0; i < 6; ++i )
    printf ("%c", word[i]);
  printf ("\n");
  return 0;
}

a special case of character arrays: the character string type => in a later chapter
Example: conversion to base b

- Convert a number from base 10 into a base b, b in range [2..16]

- Example: convert number from base 10 into base b=2

<table>
<thead>
<tr>
<th>Number</th>
<th>Number % 2</th>
<th>Number / 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step1: 10</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Step2: 5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Step3: 2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Step4: 1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Example: Base conversion using arrays

// Program to convert a positive integer to another base
#include <stdio.h>
int main (void)
{
    const char baseDigits[16] = {
        '0', '1', '2', '3', '4', '5', '6', '7',
        '8', '9', 'A', 'B', 'C', 'D', 'E', 'F' 
    };
    int convertedNumber[64];
    long int numberToConvert;
    int nextDigit, base, index = 0;
    // get the number and the base
    printf ("Number to be converted? ");
    scanf ("%ld", &numberToConvert);
    printf ("Base? ");
    scanf ("%i", &base);
// convert to the indicated base
do {
    convertedNumber[index] = numberToConvert % base;
    ++index;
    numberToConvert = numberToConvert / base;
}
while ( numberToConvert != 0 );
// display the results in reverse order
printf ("Converted number = ");
for (--index; index >= 0; --index ) {
    nextDigit = convertedNumber[index];
    printf ("%c", baseDigits[nextDigit]);
}
printf ("\n");
return 0;
}
Multidimensional arrays

- C language allows arrays of any number of dimensions
- Two-dimensional array: matrix

```c
int M[4][5]; // matrix, 4 rows, 5 columns
M[i][j] - element at row i, column j

int M[4][5] = {
    { 10, 5, -3, 17, 82 },
    { 9, 0, 0, 8, -7 },
    { 32, 20, 1, 0, 14 },
    { 0, 0, 8, 7, 6 }
};
int M[4][5] = { 10, 5, -3, 17, 82, 9, 0, 0, 8, -7, 32, 20, 1, 0, 14, 0, 0, 8, 7, 6 };```
Example: Typical matrix processing

```c
#define N 3
#define M 4

int main(void) {
    int a[N][M];
    int i, j;
    /* read matrix elements */
    for(i = 0; i < N; i++)
        for(j = 0; j < M; j++) {
            printf("a[%d][%d] = ", i, j);
            scanf("%d", &a[i][j]);
        }
    /* print matrix elements */
    for(i = 0; i < N; i++) {
        for(j = 0; j < M; j++)
            printf("%5d", a[i][j]);
        printf("\n");
    }
    return 0;
}
```
Example: Dealing with variable numbers of elements

```c
#include <stdio.h>
#define NMAX 4

int main(void) {
    int a[NMAX];
    int n;
    int i;
    printf("How many elements(maximum %d)?\n", NMAX);
    scanf("%d", &n);
    if (n > NMAX) {
        printf("Number too big !\n");
        return 1;
    }
    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);
    for (i = 0; i < n; i++)
        printf("%5d", a[i]);
    printf("\n");
    return 0;
}
```
Variable length arrays

• A feature introduced by C99
• It was NOT possible in ANSI C!
• int a[n];
• The array a is declared to contain n elements. This is called a variable length array because the size of the array is specified by a variable and not by a constant expression.
• The value of the variable must be known at runtime when the array is created => the array variable will be declared later in the block of the program
• Possible in C99: variables can be declared anywhere in a program, as long as the declaration occurs before the variable is first used.
• A similar effect of variable length array could be obtained in ANSI C using dynamic memory allocation to allocate space for arrays while a program is executing.
Example: Variable length arrays

```
#include <stdio.h>

int main(void) {
    int n;
    int i;

    printf("How many elements do you have ? \n");
    scanf("%d", &n);

    int a[n];
    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);
    for (i = 0; i < n; i++)
        printf("%5d", a[i]);
    printf("\n");
    return 0;
}
```

Array a of size n created. Value of n must be set at runtime before arriving at the array declaration!
Example: variable length arrays

```c
#include <stdio.h>

int main(void) {
    int n;
    int i;

    n=7;
    int a[n];
    for(i = 0; i < n; i++)
        a[i]=i

    n=20;
    for(i = 0; i < n; i++)
        a[i]=2*i;
    printf("\n");
    return 0;
}
```

Array a created of size 7

Variable-length array does NOT mean that you can modify the length of the array after you create it! Once created, a VLA keeps the same size!