Lecture 4: Overview

• Data Structures
  • Introduction

• Arrays
  • Introduction
  • Indexing
  • Initialization
  • Multi-dimensional arrays
  • Operator associativity and precedence

• Example
  • Histogram.c
Lecture 4: Overview (cont)

• Passing arguments to functions
  • Pass by value
  • Pass by reference
• Character Arrays: Strings
  • Input and output
  • ASCII table
  • Example: Sort strings alphabetically
    • Task
    • Approach
    • Algorithm Sort
    • Program Sort.c
Data Structures

• Introduction
  • Until now, we have used (mostly) single data elements of basic (non-composite) type
    • integral types
    • floating point types
  • Most programs, however, require complex *data structures* using composite types
    • arrays, lists, queues, stacks
    • trees, graphs
    • dictionaries
  • ANSI C provides built-in support for
    • Arrays
    • Structures, unions, enumerators
    • Pointers
Arrays

- Array data type in C
  - Composite data type
    - Type is an array of a sub-type (e.g. array of int)
  - Fixed number of elements
    - Array size is fixed at time of definition (e.g. 100 elements)
  - Element access by index (aka. subscript)
    - Element-access operator: `array[index]` (e.g. `A[42]`)
- Example:

```c
int A[10]; /* array of ten integers */
A[0] = 42; /* access to elements */
A[1] = 100;
```
Arrays

- Array Indexing
  - Start counting from 0
  - First element has index 0
  - Last element has index Size-1
- Example:

```c
int A[10];
A[0] = 42;
A[1] = 100;
/* ... */
```
Arrays

• Array Indexing
  • `for` loops are often very helpful
    • `for(i=0; i<N; i++)`
      {}

• Example:

```c
int A[10];
int i;

for(i=0; i<10; i++)
  { A[i] = i*10 + i; }
for(i=0; i<10; i++)
  { printf("%d, ", A[i]); }
```

0, 11, 22, 33, 44, 55, 66, 77, 88, 99,
Arrays

- Array Indexing
  - Array indices are *not* checked by the compiler!
  - Accessing an array with an *index out of range* results in unpredictable behavior!

- Example:

```c
int A[10];
int i;

A[-1] = 42; /* INVALID ACCESS! */

for(i=0; i<=10; i++)
  /* INVALID LOOP RANGE! */
  { printf("%d, ", A[i]);
  }
```
Arrays

- Array Initialization
  - Static initialization at time of array definition
  - Initial elements listed in `{ }`
- Example:

```c
int A[10] = { 42, 100, 310, 44, 55, 0, 3, 4, 0, 99};
```
Arrays

- Array Initialization
  - Static initialization at time of array definition
  - Initial elements listed in `{ }`
- Example:

  ```
  int A[] = { 42, 100, 310, 44, 55, 0, 3, 4, 0, 99 };
  ```

- With given initializer list, array size may be omitted
  - automatically determined
Arrays

- Array Initialization
  - Static initialization at time of array definition
  - Initial elements listed in `{ }`
- Example:
  ```
  int A[10] = { 1, 2, 3};
  ```
  - With given initializer list and array size, unlisted elements are zero-initialized
    - array is filled up with zeros
Arrays

- Multi-dimensional Arrays
  - *Array of an array*
  - Example:

```c
int M[3][2] = {{1, 2},
               {3, 4},
               {5, 6}};
int i, j;
for(i=0; i<3; i++)
  for(j=0; j<2; j++)
    printf("%d ",
           M[i][j]);
printf("\n");
```

```

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
```
Arrays

- Operator associativity and precedence
  - parentheses, array access \((\ ), [\ )\) left to right
  - unary operators \(+, -, !, ++, --\) right to left
  - type casting \((typename)\) right to left
  - multiplication, division, modulo \(*, /, \%\) left to right
  - addition, subtraction \(+, -\) left to right
  - shift left, shift right \(<<, >>\) left to right
  - relational operators \(<, <=, >=, >\) left to right
  - equality \(==, !=\) left to right
  - logical and \(&&\) left to right
  - logical or \(||\) left to right
  - conditional operator \(?:\) left to right
  - assignment operators \(=, +=, *=,\) etc. right to left
  - comma operator \(,\) left to right
Passing Arguments to Functions

• Pass by Value
  • only the *current value* is passed as argument
  • the parameter is a *copy* of the argument
  • changes to the parameter *do not* affect the argument

• Pass by Reference
  • a *reference* to the object is passed as argument
  • the parameter is a *reference* to the argument
  • changes to the parameter *do* affect the argument

• In ANSI C, ...
  • ... basic types are passed by value
  • ... arrays are passed by reference
Passing Arguments to Functions

- **Example: Pass by Value**

```c
void f(int p)
{
    printf("p before modification is \%d\n", p);
    p = 42;
    printf("p after modification is \%d\n", p);
}

int main(void)
{
    int a = 0;
    printf("a before function call is \%d\n", a);
    f(a);
    printf("a after function call is \%d\n", a);
}
```

<table>
<thead>
<tr>
<th>a before function call is 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>p before modification is 0</td>
</tr>
<tr>
<td>p after modification is 42</td>
</tr>
<tr>
<td>a after function call is 0</td>
</tr>
</tbody>
</table>

Changes to the parameter do not affect the argument!
Passing Arguments to Functions

• Example: Pass by Reference

```c
void f(int p[2])
{
    printf("p[1] before modification is %d\n", p[1]);
    p[1] = 42;
    printf("p[1] after modification is %d\n", p[1]);
}

int main(void)
{
    int a[2] = {0, 0};
    printf("a[1] before function call is %d\n", a[1]);
    f(a);
    printf("a[1] after function call is %d\n", a[1]);
}
```

Changes to the parameter do affect the argument!
Character Arrays: Strings

- Text is represented by character arrays (aka. *strings*)
  - Strings are null-terminated arrays of characters
  - String output
    - `printf()` conversion: “%s”
- Example:

```c
char s1[] = {'H', 'e', 'l', 'l', 'o', 0};
printf("s1 is %s.\n", s1);
```

`s1 is Hello.`
Character Arrays: Strings

- Text is represented by character arrays (aka. *strings*)
  - Strings are null-terminated arrays of characters
  - String output
    - `printf()` conversion: `"%s"`
- Example:

```c
char s1[] = {'H', 'e', 'l', 'l', 'o', 0};
char s2[] = "Hello";
printf("s1 is %s.\n", s1);
printf("s2 is %s.\n", s2);
```

```
s1 is Hello.
s2 is Hello.
```
Character Arrays: Strings

- Text is represented by character arrays (aka. *strings*)
  - Strings are null-terminated arrays of characters
  - String output
    - `printf()` conversion: “%s”
- Example:

```c
char s1[] = {'H', 'e', 'l', 'l', 'o', 0};
char s2[] = "Hello";
printf("s1 is %s.\n", s1);
printf("s2 is %s.\n", s2);
s1[1] = 'i';
s1[2] = 0;
printf("Modified s1 is %s.\n", s1);
```

s1 is Hello.
s2 is Hello.
Modified s1 is Hi.
Character Arrays: Strings

- Text is represented by character arrays (aka. *strings*)
  - Strings are null-terminated arrays of characters
  - String input
    - `scanf()` conversion: "%Ns",
      where N specifies maximum field width = array size - 1
    - address argument can be &string[0]

```
char s1[6];
printf("Enter a string: ");
scanf("%5s", &s1[0]);
printf("s1 is %s\n", s1);
```

Enter a string: Test
s1 is Test.
Character Arrays: Strings

- Text is represented by character arrays (aka. strings)
  - Strings are null-terminated arrays of characters
  - String input
    - `scanf()` conversion: “%Ns”, where N specifies maximum field width = array size - 1
    - address argument can be `&string[0]`

  ```c
  char s1[6];
  printf("Enter a string: ");
  scanf("%5s", s1);
  printf("s1 is %s.\n", s1);
  ```

  Enter a string: Test
  s1 is Test.
Character Arrays: Strings

- Text is represented by character arrays (aka. *strings*)
  - Strings are null-terminated arrays of characters
  - Characters are represented by numeric values
  - ASCII table defines character values 0-127
- Example:

```c
char s1[] = "ABC12";
int i = 0;
while(s1[i])
    { printf("%c = %d\n",s1[i],s1[i]);
      i++; }
```

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>s1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>1</td>
<td>B</td>
<td>2</td>
<td>C</td>
<td>'A'</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'B'</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'C'</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>'1'</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td>'2'</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

A = 65
B = 66
C = 67
1 = 49
2 = 50
# Character Arrays: Strings

- **ASCII Table**
- American Standard Code for Information Interchange

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NUL</td>
<td>1</td>
<td>SOH</td>
<td>2</td>
<td>STX</td>
<td>3</td>
<td>ETX</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>BS</td>
<td>9</td>
<td>HT</td>
<td>10</td>
<td>NL</td>
<td>11</td>
<td>VT</td>
<td>12</td>
</tr>
<tr>
<td>16</td>
<td>DLE</td>
<td>17</td>
<td>DC1</td>
<td>18</td>
<td>DC2</td>
<td>19</td>
<td>DC3</td>
<td>20</td>
</tr>
<tr>
<td>24</td>
<td>CAN</td>
<td>25</td>
<td>EM</td>
<td>26</td>
<td>SUB</td>
<td>27</td>
<td>ESC</td>
<td>28</td>
</tr>
<tr>
<td>32</td>
<td></td>
<td>33</td>
<td>!</td>
<td>34</td>
<td>&quot;</td>
<td>35</td>
<td>#</td>
<td>36</td>
</tr>
<tr>
<td>40</td>
<td>(</td>
<td>41</td>
<td>)</td>
<td>42</td>
<td>*</td>
<td>43</td>
<td>+</td>
<td>44</td>
</tr>
<tr>
<td>48</td>
<td>0</td>
<td>49</td>
<td>1</td>
<td>50</td>
<td>2</td>
<td>51</td>
<td>3</td>
<td>52</td>
</tr>
<tr>
<td>56</td>
<td>8</td>
<td>57</td>
<td>9</td>
<td>58</td>
<td>:</td>
<td>59</td>
<td>;</td>
<td>60</td>
</tr>
<tr>
<td>64</td>
<td>@</td>
<td>65</td>
<td>A</td>
<td>66</td>
<td>B</td>
<td>67</td>
<td>C</td>
<td>68</td>
</tr>
<tr>
<td>72</td>
<td>H</td>
<td>73</td>
<td>I</td>
<td>74</td>
<td>J</td>
<td>75</td>
<td>K</td>
<td>76</td>
</tr>
<tr>
<td>80</td>
<td>P</td>
<td>81</td>
<td>Q</td>
<td>82</td>
<td>R</td>
<td>83</td>
<td>S</td>
<td>84</td>
</tr>
<tr>
<td>88</td>
<td>X</td>
<td>89</td>
<td>Y</td>
<td>90</td>
<td>Z</td>
<td>91</td>
<td>[</td>
<td>92</td>
</tr>
<tr>
<td>96</td>
<td>`</td>
<td>97</td>
<td>a</td>
<td>98</td>
<td>b</td>
<td>99</td>
<td>c</td>
<td>100</td>
</tr>
<tr>
<td>104</td>
<td>h</td>
<td>105</td>
<td>i</td>
<td>106</td>
<td>j</td>
<td>107</td>
<td>k</td>
<td>108</td>
</tr>
<tr>
<td>112</td>
<td>p</td>
<td>113</td>
<td>q</td>
<td>114</td>
<td>r</td>
<td>115</td>
<td>s</td>
<td>116</td>
</tr>
<tr>
<td>120</td>
<td>x</td>
<td>121</td>
<td>y</td>
<td>122</td>
<td>z</td>
<td>123</td>
<td>{</td>
<td>124</td>
</tr>
<tr>
<td>127</td>
<td>DEL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Character Arrays: Strings

• Case Study: *Sort*
  • Task: Sort an array of strings alphabetically
  • Input: Array of 10 strings entered by the user
  • Output: Array of 10 strings in alphabetical order

• Approach: Divide and Conquer
  • Step 1: Let user enter 10 strings
  • Step 2: Sort the array of strings
  • Step 3: Output the strings in order
Character Arrays: Strings

• Case Study: Sort
  • Task: Sort an array of strings alphabetically
  • Input: Array of 10 strings entered by the user
  • Output: Array of 10 strings in alphabetical order
• Approach: Divide and Conquer
  • Step 1: Let user enter 10 strings
  • Step 2: Sort the array of strings
    • Algorithm
      - compare all possible pairs of strings and swap the pair if they are not in alphabetical order
    • String comparison
      - compare character pairs alphabetically: use ASCII table!
    • String swap (exchange two strings in place)
      - swap each character pair in the two strings
  • Step 3: Output the strings in order
Character Arrays: Strings

- **Program example: Sort.c (part 1/7)**

```c
/* Sort.c: sort strings alphabetically */
/* author: Rainer Doemer */
/* */
/* modifications: */
/* 11/06/04 RD initial version */

#include <stdio.h>

/* constant definitions */

#define NUM 10 /* ten strings */
#define LEN 20 /* of length 20 */

/* function declarations */

void EnterText(char Text[NUM][LEN]);
void PrintText(char Text[NUM][LEN]);
int CompareStrings(char s1[LEN], char s2[LEN]);
void SwapStrings(char s1[LEN], char s2[LEN]);
void Sort(char Text[NUM][LEN]);
...
```
Character Arrays: Strings

- Program example: `BubbleSort.c` (part 2/7)

```c
...  
/* function definitions */

/* let the user enter the text array */

void EnterText(char Text[NUM][LEN])
{
    int i;

    for(i = 0; i < NUM; i++)
        { printf("Enter text string %2d: ", i+1);
            scanf("%19s", Text[i]);
        } /* rof */
} /* end of EnterText */

...
Character Arrays: Strings

• Program example: BubbleSort.c (part 3/7)

```c
/* print the text array on the screen */

void PrintText(char Text[NUM][LEN])
{
    int i;

    for(i = 0; i < NUM; i++)
        { printf("String %2d: %s\n", i+1, Text[i]);
        } /* rof */
} /* end of PrintText */
```

...
Character Arrays: Strings

- Program example: BubbleSort.c (part 4/7)

```c
/* alphabetically compare strings s1 and s2: */
/* return -1, if string s1 < string s2 */
/* return  0, if string s1 = string s2 */
/* return  1, if string s1 > string s2 */

int CompareStrings(char s1[LEN], char s2[LEN])
{
    int i;

    for(i = 0; i < LEN; i++)
    {
        if (s1[i] > s2[i])
        { return(1); }
        if (s1[i] < s2[i])
        { return(-1); }
        if (s1[i] == 0 || s2[i] == 0)
        { break; }
    } /* rof */
    return 0;
} /* end of CompareStrings */
...
Character Arrays: Strings

• Program example: BubbleSort.c (part 5/7)

```c
/* swap/exchange the strings s1 and s2 in place */

void SwapStrings(char s1[LEN], char s2[LEN])
{
    int  i;
    char c;

    for(i = 0; i < LEN; i++)
        { c = s1[i];
          s1[i] = s2[i];
          s2[i] = c;
        } /* rof */
} /* end of SwapStrings */
```

...
Character Arrays: Strings

- Program example: **BubbleSort.c** (part 6/7)

```c
/* sort the text array by comparing every pair */
/* of strings; if the pair of strings is not in */
/* alphabetical order, swap it */

void Sort(char Text[NUM][LEN])
{
    int i, j;

    for(i = 0; i < NUM-1; i++)
    {
        for(j = i+1; j < NUM; j++)
        {
            if (CompareStrings(Text[i], Text[j]) > 0)
            {
                SwapStrings(Text[i], Text[j]);
            }
        }
    }
}
/* end of BubbleSort */
```
Character Arrays: Strings

- Program example: BubbleSort.c (part 7/7)

```c
.../* main function: enter, sort, print the text */
int main(void)
{
   /* local variables */
   char Text[NUM][LEN]; /* NUM strings, length LEN */
   /* input section */
   EnterText(Text);
   /* computation section */
   Sort(Text);
   /* output section */
   PrintText(Text);
   /* exit */
   return 0;
} /* end of main */
/* EOF */
```
Character Arrays: Strings

- Example session: `Sort.c`

```
% vi Sort.c
% gcc Sort.c -o Sort -Wall -ansi
% Sort
Enter text string  1: Charlie
Enter text string  2: William
Enter text string  3: Donald
Enter text string  4: John
Enter text string  5: Jane
Enter text string  6: Jessie
Enter text string  7: Donald
Enter text string  8: Henry
Enter text string  9: George
Enter text string 10: Emily
String  1: Charlie
String  2: Donald
String  3: Donald
String  4: Emily
String  5: George
String  6: Henry
String  7: Jane
String  8: Jessie
String  9: John
String 10: William
%```
Arrays

- Array data type in C
  - Composite data type
    - Type is an array of a sub-type (e.g. array of `int`)
  - Fixed number of elements
    - Array size is fixed at time of definition (e.g. 100 elements)
  - Element access by index (aka. subscript)
    - Element-access operator: `array[index]` (e.g. `A[42]`)
- Example:

```c
int A[10]; /* array of ten integers */
A[0] = 42; /* access to elements */
A[1] = 100;
```
Arrays

- Array Indexing
  - Start counting from 0
    - First element has index 0
    - Last element has index Size-1
  - Example:

```c
int A[10];
A[0] = 42;
A[1] = 100;
/* ... */
```
Arrays

• Array Indexing
  • Array indices are *not* checked by the compiler!
  • Accessing an array with an *index out of range* results in unpredictable behavior!

• Example:

```c
int A[10];
int i;

A[-1] = 42; /* INVALID ACCESS! */

for(i=0; i<=10; i++)
    /* INVALID LOOP RANGE! */
    { printf("%d, ", A[i]);
    }
```
Arrays

- Array Initialization
  - Static initialization at time of array definition
  - Initial elements listed in \{ \}
- Example:

```c
int A[10] = { 42, 100, 310, 44, 55, 0, 3, 4, 0, 99};
```
Arrays

- Array Initialization
  - Static initialization at time of array definition
  - Initial elements listed in \{ \}
- Example:

```c
int A[ ] = { 42, 100, 310, 44, 55, 0, 3, 4, 0, 99};
```

- With given initializer list, array size may be omitted
  - automatically determined
Arrays

- Array Initialization
  - Static initialization at time of array definition
  - Initial elements listed in `{ }`
  
Example:

```c
int A[10] = { 1, 2, 3};
```

- With given initializer list and array size, unlisted elements are zero-initialized
  - array is filled up with zeros
Arrays

- Multi-dimensional Arrays
  - *Array of an array...*
- Example:

```c
int M[3][2] = {{1, 2},
               {3, 4},
               {5, 6}};

int i, j;

for(i=0; i<3; i++)
    for(j=0; j<2; j++)
        printf("%d ",
               M[i][j]);

printf("\n");
```

```
M | 0 | 1
---|---|---
0 | 1 | 2
1 | 3 | 4
2 | 5 | 6
```

1 2 3 4 5 6
Passing Arguments to Functions

• Pass by Value
  • only the current value is passed as argument
  • the parameter is a copy of the argument
  • changes to the parameter do not affect the argument

• Pass by Reference
  • a reference to the object is passed as argument
  • the parameter is a reference to the argument
  • changes to the parameter do affect the argument

• In ANSI C, ...
  • ... basic types are passed by value
  • ... arrays are passed by reference
Passing Arguments to Functions

• **Example: Pass by Value**

```c
void f(int p)
{
    printf("p before modification is %d\n", p);
    p = 42;
    printf("p after modification is %d\n", p);
}

int main(void)
{
    int a = 0;
    printf("a before function call is %d\n", a);
    f(a);
    printf("a after function call is %d\n", a);
}
```

| a before function call is 0 | p before modification is 0 | p after modification is 42 | a after function call is 0 |

Changes to the parameter do not affect the argument!
Passing Arguments to Functions

• Example: Pass by Reference

```c
void f(int p[2])
{
    printf("p[1] before modification is %d\n", p[1]);
    p[1] = 42;
    printf("p[1] after modification is %d\n", p[1]);
}

int main(void)
{
    int a[2] = {0, 0};
    printf("a[1] before function call is %d\n", a[1]);
    f(a);
    printf("a[1] after function call is %d\n", a[1]);
}
```

Changes to the parameter do affect the argument!
Text is represented by character arrays (aka. *strings*)

- Strings are null-terminated arrays of characters
- String output
  - `printf()` conversion: \texttt{“%s”}
- Example:

```c
char s1[] = {'H', 'e', 'l', 'l', 'o', 0};
printf("s1 is %s.\n", s1);
```

s1 is Hello.
Character Arrays: Strings

- Text is represented by character arrays (aka. *strings*)
  - Strings are null-terminated arrays of characters
  - String output
    - `printf()` conversion: “%s”
- Example:

```c
char s1[] = {'H','e','l','l','o',0};
char s2[] = "Hello";
printf("s1 is %s.\n", s1);
printf("s2 is %s.\n", s2);
```

**s1** is Hello.
**s2** is Hello.
Character Arrays: Strings

- Text is represented by character arrays (aka. *strings*)
  - Strings are null-terminated arrays of characters
  - String output
    - `printf()` conversion: "%s"
- Example:

```c
char s1[] = {'H', 'e', 'l', 'l', 'o', 0};
char s2[] = "Hello";
printf("s1 is %s.\n", s1);
printf("s2 is %s.\n", s2);
s1[1] = 'i';
s1[2] = 0;
printf("Modified s1 is %s.\n", s1);
```

s1 is Hello.
s2 is Hello.
Modified s1 is Hi.
Character Arrays: Strings

- Text is represented by character arrays (aka. *strings*)
  - Strings are null-terminated arrays of characters
  - String input
    - `scanf()` conversion: “%Ns”, where N specifies maximum field width = array size - 1
    - address argument can be `&string[0]`

```c
char s1[6];
printf("Enter a string: ");
scanf("%5s", &s1[0]);
printf("s1 is %s.\n", s1);
```

Enter a string: Test
s1 is Test.
Character Arrays: Strings

• Text is represented by character arrays (aka. *strings*)
  • Strings are null-terminated arrays of characters
  • String input
    • `scanf()` conversion: “`%Ns`”, where \( N \) specifies maximum field width = array size - 1
    • address argument can be `&string[0]`

```
char s1[6];
printf("Enter a string: ");
scanf("%5s", s1);
printf("s1 is %s.\n", s1);
```

Enter a string: Test
s1 is Test.
Character Arrays: Strings

- Text is represented by character arrays (aka. *strings*)
  - Strings are null-terminated arrays of characters
  - Characters are represented by numeric values
  - ASCII table defines character values 0-127

- Example:
  ```c
  char s1[] = "ABC12";
  int i = 0;
  while(s1[i])
    { printf("%c = %d\n",s1[i],s1[i]);
      i++;
    }
  
  A = 65
  B = 66
  C = 67
  1 = 49
  2 = 50
  ```