Programming Fundamentals

Module F - Arrays

Le The Anh
anhlt161@fe.edu.vn

FPT UNIVERSITY

Objectives

1. Array
   - Class Problem (without array)
   - Array Definition
   - Elements
   - Initialization
   - Passing Arrays to Functions
   - Solution for Class Problem (with array)
2. Parallel Arrays
   - Tabular Information
   - Record Searches
   - In-Class Practice
   - Exercises
3. Mixing, Masking and Sorting Algorithms
   - Mixing Algorithms
   - Masking
   - In-Class Practice
   - Sorting Algorithms

Simple Data Structures

- A data structure is a collection of data types designed to store information in some optimal way.
- Data structures improve readability and simplify coding considerably.
- The simplest example of a data structure is an array.

Class Problem (without array)

- Task: Design and code a program that displays the binary equivalent of a positive integer less than 32767.

Class Problem (without array)

- Task: Design and code a program that displays the binary equivalent of a positive integer less than 32767.
- Consider one solution using pointer on the reference website.
- Any other solution?
- What are the drawbacks of discussed solutions?

Array Definition

Definition

An array is an ordered set of related elements of common data type that are stored contiguously in memory.

- Declaration syntax

```
dataType identifier[size];
```

- Example

```
// allocates contiguous storage in primary memory
// for an array named digit of 16 int elements
int digits[16];

// check how the elements are stored in the memory
for(i=0; i<16; i++)
    printf("%p\n", &digits[i]);
```

Array Definition

- A constant can be used to specify the size of the array

```c
#include <stdio.h>
#define N_BITS 16

main()
{
    // define an array of N_BITS integer elements
    int digits[N_BITS];
    int i;

    // check how the elements are stored in the memory
    for(i=0; i<N_BITS; i++)
        printf("%p\n", &digits[i]);
}
```

Array Elements

- The fundamental unit of an array is an element.
- Each element has an index and holds one data value.
- Index numbering starts at 0 and extends to one less than the size of the array.
- To refer to an element, we use the array name followed by bracket notation around the element index. For example,

```
// print the fourth element
printf("%d", digits[3]);
```

- The index may be an integer variable, an integer expression or an integer constant. For example,

```
int i = 3;
printf("%d", digits[i]);
```

Initialization

```
Syntax: dataType arrayIdentifier[size] = {value, value, ..., value};

// don't need to specify the size if all elements are initialized
int a[] = {1, 2, 3, 4, 5};
// if the specified size is greater than number of initial values
// the uninitialized elements will be filled with zeros
int b[5] = {1, 2};
int i;

for(i=0; i<5; i++)
    printf("%d ", a[i]);

    printf("\n");

for(i=0; i<5; i++)
    printf("%d ", b[i]);

/********
1 2 3 4 5
1 2 0 0 0
/********
```

Passing Arrays to Functions

- Because compilers store the elements of an array contiguously in memory, passing the address of the array to a function is sufficient to enable access to all of the elements of the array within the function.
- The location of any element is simply the produce of the offset from the address of the first element and the number of bytes in a single element.
- By only passing the address of the array, we avoid copying the array, which may be a rather expensive operation.

Passing Arrays to Functions

- The syntax of a function header that receives an array address is

```
dataType functionIdentifier (dataType arrayIdentifier[], ...)
```

```
// note that arrays are always passed as reference
void dec2bin(decimal, char digits[])
{
    // do something
}
```

- The syntax of a function call that passes an array is

```
functionIdentifier (arrayIdentifier, ...)
```

```
dec2bin(n, digits);
```

Solution for Class Problem (with array)

- Refer the solution for the class problem on the reference website.

Summary

- Array
  - Class Problem (without array)
  - Array Definition
  - Elements
  - Initialization
  - Passing Arrays to Functions
  - Solution for Class Problem (with array)

Q&A

In-Class Practice

1. Write a program to find the maximum odd number in a given list of integer numbers.
2. Write a program to find the longest increasing contiguous sub-array in a given array.

Tabular Information

- Parallel arrays are arrays of identical size that hold elements in a common order. The i-th element of one array is related to the i-th element of a parallel array.
- Parallel arrays store tabular information

sku	Unit Price
1234	12.34
2345	13.12
3456	45.23

Tabular Information

```c
#include <stdio.h>
#define MAX_ITEMS 20

int main ( ) {
    int i, nItems, keepgoing, s, sku[MAX_ITEMS];
    double unitPrice[MAX_ITEMS];

    printf("Enter skus and unit prices\n");
    nItems = 0;
    keepgoing = 1;
    do {
        printf("SKU (0 to stop) : ");
        scanf("%d", &s);
        if (s != 0) {
            sku[nItems] = s;
            printf("Unit Price : ");
            scanf("%lf", &unitPrice[nItems]);
            nItems++;
        } else
            keepgoing = 0;
    } while ( keepgoing == 1 && nItems < MAX_ITEMS );

    printf("SKU    Unit Price\n");
    for (i = 0; i < nItems; i++)
         printf("%06d $%9.2lf\n", sku[i], unitPrice[i]);

    return 0;
}
```

Record Searches

- In a set of parallel arrays, one array can serve to hold a unique record identifier. We call that array the key array. A search algorithm finds the record of interest using the key array.
- Two common search algorithms: linear search, binary search.

Linear Search

- A linear search algorithm starts at one end of the key array and steps through the elements sequentially, until the algorithm locates the search value.

```
// find returns the first index where search == key[index]
// for key[size], or -1 if no match found
int find(int search, int key[], int size)
{
    int i = 0, rc = -1;

    for (i = 0; i < size && rc == -1; i++)
        if (search == key[i])
            rc = i;

    return rc;
}
```

Binary Search

- A binary search algorithm works with a sorted array.
- The algorithm discards half of the array that does not contain the search value and keeps discarding the half of the remaining subset of elements that does not contain the search value.

Binary Search

```
// binaryFind returns the first index where
// search == key[index] for key[size] where
// key is sorted in ascending order, or -1 if match not found
int binaryFind(int search, int key[], int size)
{
    int rc = -1, i, low = 0, high = size-1;

    do
    {
        // calculate mid-element
        i = (low + high + 1) / 2;
        if (search < key[i])
            // reset high element
            high = i - 1;
        else if (search > key[i])
            // reset low element
            low = i + 1;
        else
            // found it
            rc = i;
    } while (rc == -1 && low <= high);

    return rc;
}
```

In-Class Practice

- Design and code a program named cable.c that
  - prompts the user for a tv station number, and
  - displays the corresponding cable channel.
- The tv-station cable-channel correspondence list is

```
TV:      2   3   4   5   6   7   9   11  17  25  29  36
Cable:   17  20  16  06  03  18  08  11  61  12  28  04
```

Exercises

- Do the exercises given on the reference website.

Mixing Algorithms

- Examples of algorithms for mixing elements include ones for shuffling the cards in a deck or tumbling numbered balls into a lottery chute.
- Such algorithms vary in the extent to which they generate a truly fair result.

Linear Shuffle

- Consider a program that shuffles a deck of cards and displays the results of the shuffle. With each iteration, the program sets aside one card and shuffles the remaining cards.

```
// linear_shuffle shuffles a deck of cards in linear time
void linear_shuffle(int card[])
{
    int i, cardsLeft, j, temp;

    cardsLeft = CARDS;
    for (i = 0; i < CARDS; i++)
    {
        j = i + rand() % cardsLeft;
        temp = card[i];
        card[i] = card[j];
        card[j] = temp;
        cardsLeft--;
    }
}
```

Masking

- Masking is a technique where one array hides or 'masks' the corresponding elements of parallel arrays.
- Consider the drawing of cards from a shuffled deck. To identify a card as having been drawn, let us introduce an array named drawn that is parallel to card[] and initialize the elements of this masking array to 0. For each card drawn, let us reset the corresponding element in drawn to 1.

Masking

```
// drawCards draws n cards from a deck
// identifying the drawn cards by setting drawn[i] to 1
void draw(int drawn[], int nDraw)
{
    int i, j, notDrawn, card, skipped, keeplooking;

    notDrawn = CARDS;
    for (i = 0; i < nDraw; i++) {
        // randomly select a card from the remaining deck to draw
        card = rand() % notDrawn;
        skipped = 0;
        keeplooking = 1;
        for (j = 0; j < CARDS && keeplooking == 1; j++)
        {
            // draw the 'card-th' card from the remaining deck
            if (drawn[j] == 0 && skipped == card)
            {
                drawn[j] = 1;
                keeplooking = 0;
            } else if (drawn[j] == 0)
                skipped++;
        }
        notDrawn--;
    }
}
```

Flagging

- Masking array can be used to find the subtotals of common valued items in a list, for example

sku	Units Sold
1234	4
2345	15
3456	9
2345	7
1234	6
3456	3
2345	3
3456	12
1234	1

$\rightarrow$

sku	Units Sold
1234	11
2345	25
3456	24

Flagging

```
// flag all entries as uncounted
for ( i = 0; i < nItems; i++ )
    accountedFor[i] = 0;

printf("SKU    Units Sold\n");
for (i = 0; i < nItems; i++)
{
    // consider only the unaccounted for elements
    if (accountedFor[i] != 1)
    {
        // this sku is next to be counted
        code = sku[i];
        total = 0;
        for (j = i; j < nItems; j++)
        {
            if (code == sku[j])
            {
                total += units[j];
                // mark j as accounted for
                accountedFor[j] = 1;
            }
        }
        printf("%06d %9d\n", code, total);
    }
}
```

In-Class Practice

- Try the practice problem in Handout 12.

Sorting Algorithms

- Two common algorithms:
  - Selection sort
  - Bubble sort

Selection Sort

- Pseudo-code:
  1. Find the maximum value in the list
  2. Swap it with the value in the last position
  3. Repeat the steps above for remainder of the list
- Example: 1 3 2 5 4
  1. 1 3 2 5 4
  2. 1 3 2 4 5
  3. 1 3 2 4 5
  4. 1 3 2 4 5
  5. 1 3 2 4 5
  6. 1 2 3 4 5
  7. 1 2 3 4 5
  8. 1 2 3 4 5
  9. 1 2 3 4 5
  10. 1 2 3 4 5

  !!! black: unsorted elements, blue: maximum value, orange: swapped elements, red: sorted elements

Selection Sort

```
// sort the elements of a[size] in ascending order
    void selection_sort(int a[], int size)
{
    int i, j, jmax, tmp;
    for (i = 0; i < size - 1; i++)
    {
        jmax = 0;
        for (j = 1; j < size - i; j++)
            if (a[j] > a[jmax])
                jmax = j;
        if (jmax != size - i - 1)
        {
            tmp = a[size - i - 1];
            a[size - i - 1] = a[jmax];
            a[jmax] = tmp;
        }
    }
}
```

Bubble Sort

- Bubble sort repeatedly steps through the list to be sorted, comparing two items at a time and swapping them if they are in the wrong order.
- The pass through the list is repeated until no swaps are needed, which means the list is sorted.
- Example: 4 5 2 1 3
  1. 4 2 5 1 3
  2. 4 2 1 5 3
  3. 4 2 1 3 5 $\rightarrow$ 4 2 1 3 5
  4. 2 4 1 3 5
  5. 2 1 4 3 5
  6. 2 1 3 4 5 $\rightarrow$ 2 1 3 4 5
  7. 1 2 3 4 5 $\rightarrow$ 1 2 3 4 5
  8. 1 2 3 4 5
  9. 1 2 3 4 5

Bubble Sort

```
// sorts the elements of a[size] in ascending order
void bubbleSort(int a[], int size)
{
    int i, j;
    int temp;

    for (i = size - 1; i > 0; i--)
    {
        for (j = 0; j < i; j++)
        {
            if (a[j] > a[j+1])
            {
                temp = a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
            }
        }
    }
}
```

Summary

- Mixing, Masking, and Sorting Algorithms
  - Mixing Algorithms
  - Masking
  - Sorting Algorithms

Q&A