Array 1 (Summarizing) Assignment

202231774 / Department of Economics / Kim Ju Kyeong

What is Array?

Collection of Variables of the Same Data Type

Ex)

int value1;

int value2;

int value3;

…

Int value1000;

The Array can be represented in this way.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| value1 | value2 | value3 | …. | …. | …. | …. | ….. | value999 | value1000 |

Many variables of the same data type to be used for the same purpose.

Ex)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| value[0] | value[1] | value[2] | …. | …. | …. | …. | ….. | value[998] | value[999] |

\*One thing to be careful about is counting from the number zero.

1-Dimensional Number Array

It is expressed in this way. -> data\_type array(variable)\_name[index];

What is index?

index : number of elements in the array

you can use the macro or constant type declared through #define for index.

Ex)

int jangbaljang[24601]; -> It is an array with a range of 24601

#define Stress 100

float midterm[Stress]; -> It is an array with a range of 100

-Index may be an integer expression

Ex) value[3\*min+20]

value[j] = value[j-2] / value[(j-1)\*3];

Expression must have value within the index limit.

What happens if run code out of range?

Ex code)

#include <stdio.h>

int main(void)

{

int a[777];

for (int i = 0; i < 888; i++)

A[i] = i;

}

->Overflow occurs.

So, when using an array, you must keep the range of the specified index.

-All Array Elements Are Stored in Contiguous Memory Locations. (not initialized)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| !@#$ | &\*@( |  | \*&^% |  | ㅡ<>? |  |  |  | $1#% |

value[0] value[1] value[999]

-Each Array Element Must Be Initialized Before Use.

Ex code)

value[0] = 33;

value[2] = value[1] + 475;

A Major Use of an Array

-Combined with a Loop, Accumulate Values of Many Variables

Ex code1)

int i, JangBalJang[24601];

for (i = 0; i < 24601; i++)

{

printf (“\n type a grade: ”);

scanf (“%d”, &JangBalJang[i]);

}

Ex code2)

int i, gaus[100];

int count, total;

total = 0; count = 100;

for (i = 0; i < count; i++)

total = total + gaus[i];

Ex code3) (finding a minimum in an array of numbers

int i, value[1000];

int count, minValue;

minValue = value[0];

count = 1000;

for (I = 1; I < count; i++)

if (value[i] < minValue)

minValue = value[i]

-array overflow & underflow

Ex) overflow

When variable declared -> int JangBalJang[24601];

If you input a number to JangBalJang[24601], JangBalJang[25555]

->overflow occurs. (when over the maximum of index)

Ex) underflow

When variable declared -> int JangBalJang[24601];

If you input a number to JangBalJang[-24601], JangBalJang[-1]

->underflow occurs. (when below zero)

-Initialization in array

Ex) int value[50] = {11, 22, 33, 44, 55};

->from value[0] to value[4] were initialized by the specified value.

And value[5] to value[49] are initialized to zero.

If) When all are initialized to zero.

Can modify the code as follows.

->int value[50] = {0, };

-Initialization in array(use #define)

Ex)

#include <stdio.h>

#define Crime\_num 50

int main(void)

{

int JangBalJang[Crime\_num] = {0, };

for (int j = 0; j < Crime\_num; j++)

JangBalJang[j] += 2;

}

-Advantages and Disadvantages of Array

Advantages : “Direct” (Random) Access to Any Element Is Possible.

Disadvantages :

- Moving Elements Is Expensive.

- Potential Memory waste : Usually a much larger memory space is allocated than necessary.

- Re-Sizing an Array Is Expensive or Not Possible. : Usually array size is fixed.

- Deleting Elements Within an Array Leaves Holes : explicit memory waste

(Continue on the next page)

2-Dimensional Array

It is expressed in this way. -> data\_type array(variable)\_name[i][j];

Also, it can be expressed in the table as follows.

(Array that can store 15 pieces of data)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0,0 | 0,1 | 0,2 | 0,3 | 0,4 |
| 1,0 | 1,1 | 1,2 | 1,3 | 1,4 |
| 2,0 | 2,1 | 2,2 | 2,3 | 2,4 |

data\_type array(variable)\_name[i][j];

i = maximum number of rows (on the table i is 3)

j = maximum number of columns (on the table j is 5)

ex)

float midterm\_score[students][tests];

Every array element is a variable.

-Assignment to a 2-Dimensional Array

The first row can be expressed as follows. -> value[0][j];

The Second column can be expressed as follows. -> value[i][1];

-Initializing a 2-Dimensional Array

Ex) int value [4][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}, {10, 11, 12}};

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |
| 10 | 11 | 12 |

2-Dimensional Array – Nested Loop

Ex)

int value [1000][100];

int rnum=1000, cnum=100;

int rx, cx;

for (rx=0; rx < rnum; rx += 1)

{

for (cx = 0; cx < cnum; cx += 1)

printf(“%d”, value[rx][cx]);

printf(“\n”);

}

->low[0][cx] (cx start zero to finish 99)

->low[1][cx] (cx start zero to finish 99)

->low[2][cx] (cx start zero to finish 99)

…

->low[999][x] (cx start zero to finish 99)

2-Dimensional Array – Initialization

Ex)

int value [1000][100];

int rnum=1000, cnum=100;

int rx, cx;

for (rx=0; rx < rnum; rx += 1)

{

for (cx = 0; cx < cnum; cx += 1)

value[rx][cx] = 0;

}

* Value[0][0] = 0, [0][1] = 0, [0][2] = 0,… , [999][99] = 0

2-Dimensional Array – find min or max

int min;

min = value[0][0]

for (rx=0; rx < rnum; rx += 1)

{

for (cx = 0; cx < cnum; cx += 1)

{

If (min > value[rx][cx])

Min = value[rx][cx];

}

2- Dimensional Array – Nested Loop – Access one row/col

Ex)

int value [1000][100];

int rnum=1000, cnum=100;

int rx, cx;

for (cx = 0; cx < cnum; cx += 1)

value[1][cx] = 0;

for (rx = 0; rx < rnum; rx += 1)

value[rx][1] = 0;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 |  |  |  |
| 0 | 0 | 0 | 0 | 0 |
|  | 0 |  |  |  |
|  | 0 |  |  |  |

-This is how it appears in the table.

2- Dimensional Array – Matrix multiplication

Multiplication between two matrices is defined by

A = l \* m <- (rows \* columns)

B = m \* n <- (rows \* columns)

C = l \* n <- (rows of A \* columns of B)

(Continue on the next page)

When constructing a matrix into codes, you must fix the variables. (use const)

Ex)

#include <stdio.h>

Int main(void)

{

const int l = 5;

const int m = 3;

const int n = 4;

int A[l][m];

int B[m][n];

int C[l][n];

for (int i = 0; i < l; i++){

for (int j = 0; j < n; j++) {

C[i][j] = 0;

for (int k = 0; k < m; k++) {

C[i][j] += (A[i][k] \* B[k][j]);

}

}

}

}