

Review on C Programming

Review on C Programming

- What should you do with the C programming?
 - How to use an **array** and a **pointer**
 - Understand the relationship between the array and the pointer.
 - Use **dynamic memory allocation**.
 - How to define a **structure** and make use of it
 - Understand **self-referential structures**.
 - How to use **recursive programming**
 - Transform iterative programming into recursive programming.
- A Programming Course in any language is a prerequisite for this course!

What is Array?

■ Definition

- A collection of elements with same data types
- Each element is sequential in memory.

```
int score[10];
```

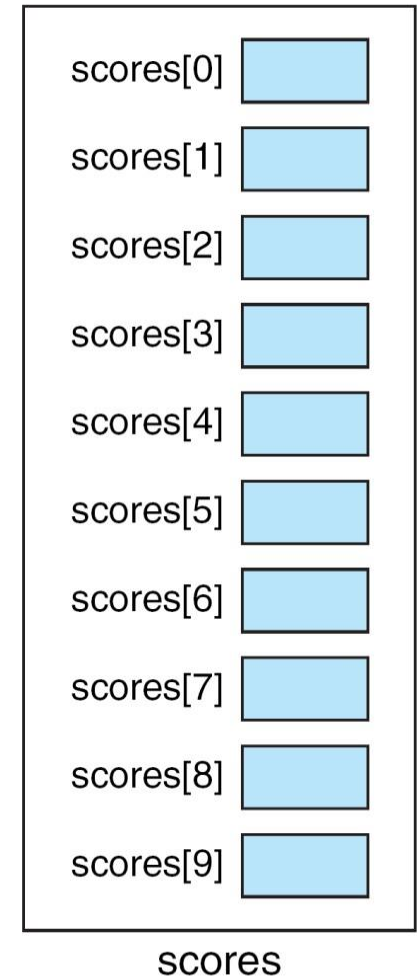
Data type

Array name

Array size

■ Array index

- Each element can be referenced by an **index**.
- **Index ranges: [0] ~ [size - 1]**



Example: Array

- Input five numbers and print them reversely.

```
#include <stdio.h>
#define ARRAY_SIZE    5

int main()
{
    int numbers[ARRAY_SIZE], i;

    printf("Input five numbers\n");
    for (i = 0; i < ARRAY_SIZE; i++)
        scanf("%d", &numbers[i]);

    for (i = ARRAY_SIZE - 1; i >= 0; i--)
        printf("%d ", numbers[i]);

    return 0;
}
```

Function Call with Array

- Calculating the average of values in array

```
#include <stdio.h>
#define ARRAY_SIZE    5

void inputNumbers(int num[], int len);
double computeAverage(int num[], int len);

int main()
{
    int numbers[ARRAY_SIZE];

    inputNumbers(numbers, ARRAY_SIZE);
    printf("average: %.3lf", computeAverage(numbers, ARRAY_SIZE));

    return 0;
}
```

Function Call with Array

- Calculating the average of values in array

```
void inputNumbers(int num[], int len)
{
    int i;
    for (i = 0; i < len; i++)
        scanf("%d", &num[i]);
}

double computeAverage(int num[], int len)
{
    int total = 0, i;
    for (i = 0; i < len; i++)
        total = total + num[i];
    return total / (double)len;
}
```

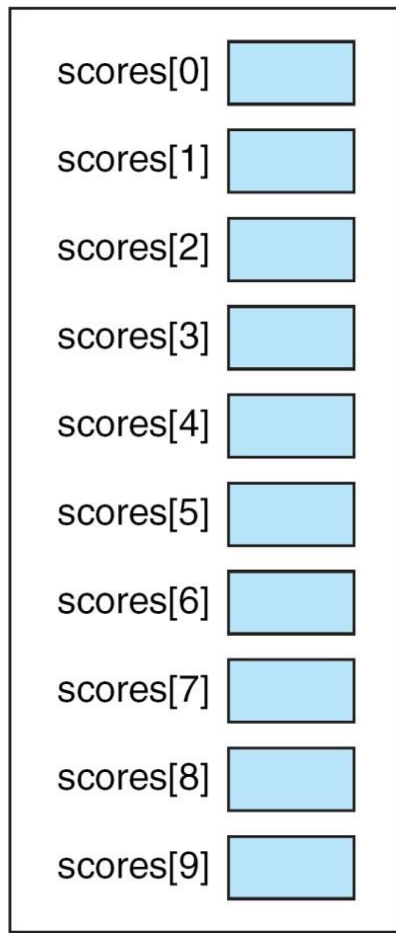
Two-Dimensional Array

- Filling all entries in the two-dimensional matrix

```
int row, col, matrix[6][6];  
  
for (row = 0; row < 6; row++)  
{  
    for (col = 0; col < 6; col++)  
    {  
        if (row < col)  
            matrix[row][col] = 1;  
        else if (row == col)  
            matrix[row][col] = 0;  
        else  
            matrix[row][col] = -1;  
    }  
}
```

0	1	1	1	1	1
-1	0	1	1	1	1
-1	-1	0	1	1	1
-1	-1	-1	0	1	1
-1	-1	-1	-1	0	1
-1	-1	-1	-1	-1	0

`int scores[10];`



`scores`

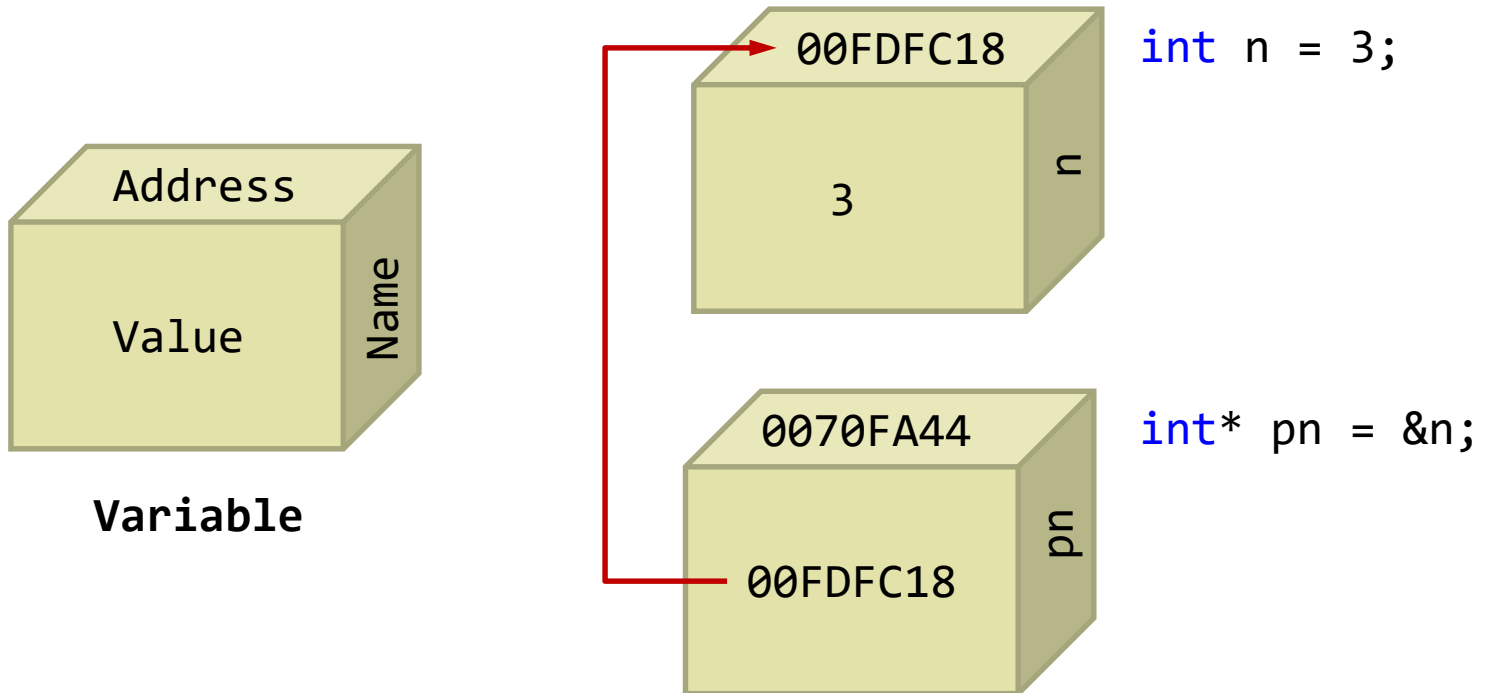
`int matrix[6][6];`

0	1	1	1	1	1
-1	0	1	1	1	1
-1	-1	0	1	1	1
-1	-1	-1	0	1	1
-1	-1	-1	-1	0	1
-1	-1	-1	-1	-1	0

What is Pointer?

■ Definition

- A variable to store a **memory address** instead of a value



& (Ampersand) Operator

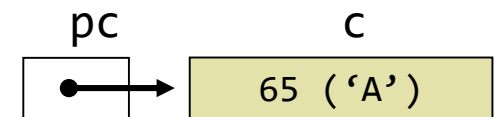
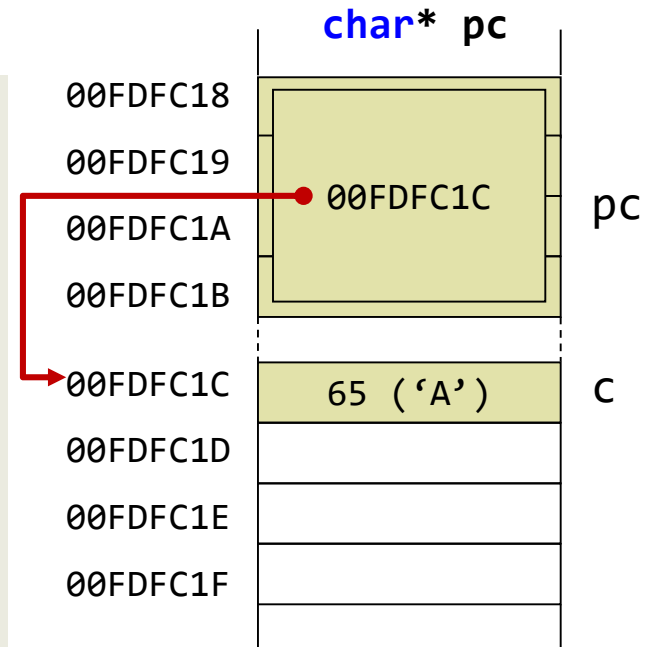
- Reference operator
 - Return the address of an variable.

```
#include <stdio.h>

int main()
{
    char c = 'A';
    char *pc = &c;

    printf("%c %p\n", c, pc);
    printf("%p %p\n", &c, &pc);
    printf("%d %d\n", sizeof(c), sizeof(pc));

    return 0;
}
```



& (Ampersand) Operator

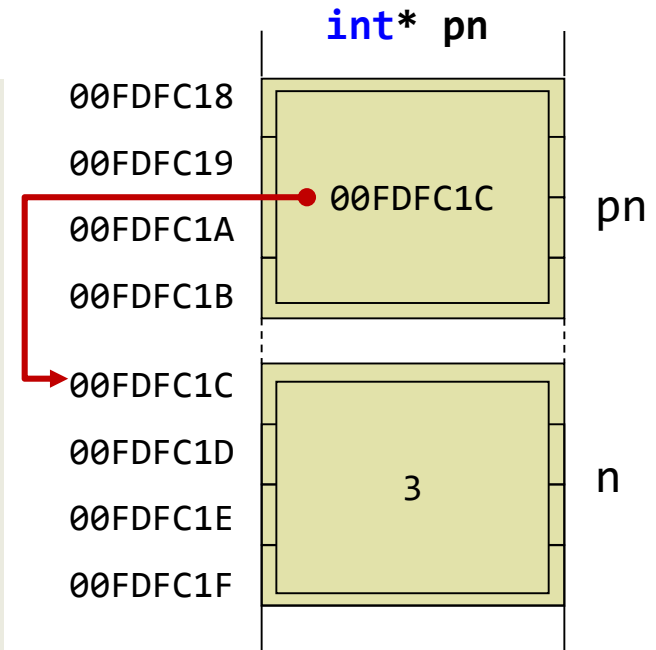
- What is the size of a pointer variable?

```
#include <stdio.h>

int main()
{
    int n = 3;
    int *pn = &n;

    printf("%d %p\n", n, pn);
    printf("%p %p\n", &n, &pn);
    printf("%d %d\n", sizeof(n), sizeof(pn));

    return 0;
}
```



* (Asterisk) Operator

- Dereference operator
 - Return the value at the pointer address.

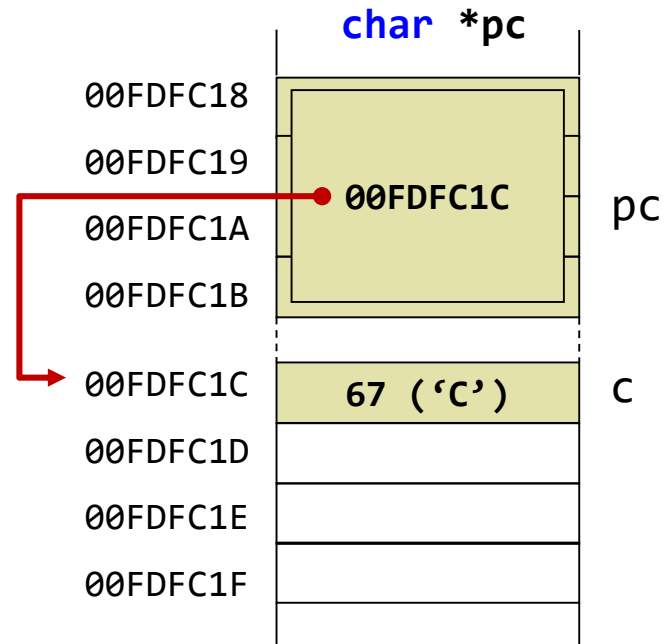
```
#include <stdio.h>

int main()
{
    char c = 'A';
    char *pc = &c;

    printf("%c %c\n", c, *pc);

    *pc = 'C';
    printf("%c %c\n", c, *pc);

    return 0;
}
```



Example: Pointer

- Use address and dereference operators correctly.

```
#include <stdio.h>

int main()
{
    int a, b, c;
    int *p, *q, *r;

    a = 6, b = 10;
    p = &b, q = p, r = &c;
    p = &a, *q = 8, *r = *p;
    *r = a + *q + *&c;

    printf("%d %d %d", a, b, c);

    return 0;
}
```

Example: Pointer

```
#include <stdio.h>

int main()
{
    int a, b, c;
    int *pa = &a, *pb = &b, *pc = &c;

    *pa = 10, *pb = 20;
    *pc = *pa + *pb;

    printf("%d %d %d", a, b, c);

    return 0;
}
```

Functional Call with Pointer

- Two types for inter-function communication
 - **Call by value:** passing by value
 - **Call by reference:** passing by address

```
#include <stdio.h>

void swap1(int x, int y);
void swap2(int* px, int* py);

int main()
{
    int a = 5, b = 7;
    swap1(a, b);
    printf("%d %d\n", a, b);
    swap2(a, b);
    printf("%d %d\n", a, b);
    return 0;
}
```

```
void swap1(int x, int y)
{
    int temp = x;
    x = y;
    y = temp;
}
```

```
void swap2(int* px, int* py)
{
    int temp = *px;
    *px = *py;
    *py = temp;
}
```

Pointer to Pointer

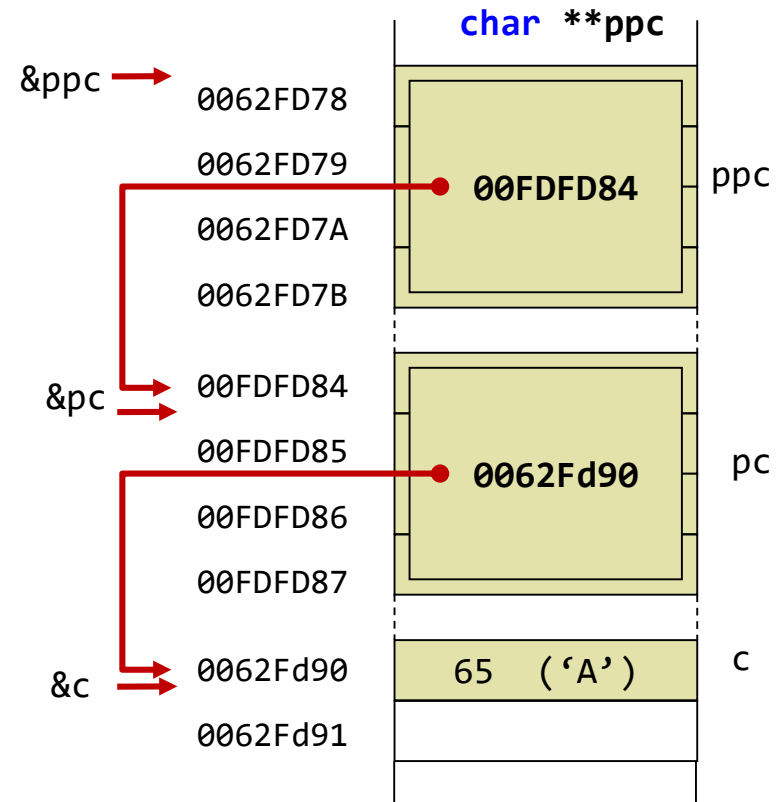
- We can use a pointer that points to another pointer.

```
#include <stdio.h>

int main()
{
    char c = 'A';
    char* pc = &c;
    char** ppc = &pc;

    printf("%p %p\n", pc, ppc);
    printf("%d %d\n", sizeof(pc), sizeof(ppc));

    return 0;
}
```



Arithmetic Operation for Pointer

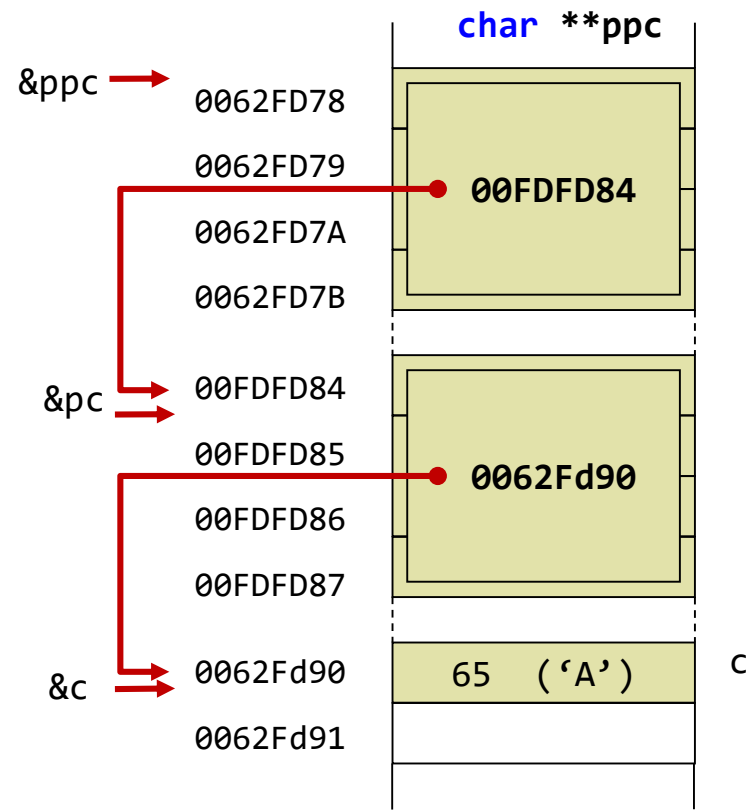
■ How does the char pointer work?

```
#include <stdio.h>

int main()
{
    char c = 'A';
    char* pc = &c;
    char** ppc = &pc;

    printf("%p %p\n", pc, ppc);
    printf("%p %p\n", pc + 1, ppc + 1);
    printf("%p %p\n", &c, &c + 1);
    printf("%p %p\n", &pc, &ppc);
    printf("%p %p\n", &pc + 1, &ppc + 1);

    return 0;
}
```



Arithmetic Operation for Pointer

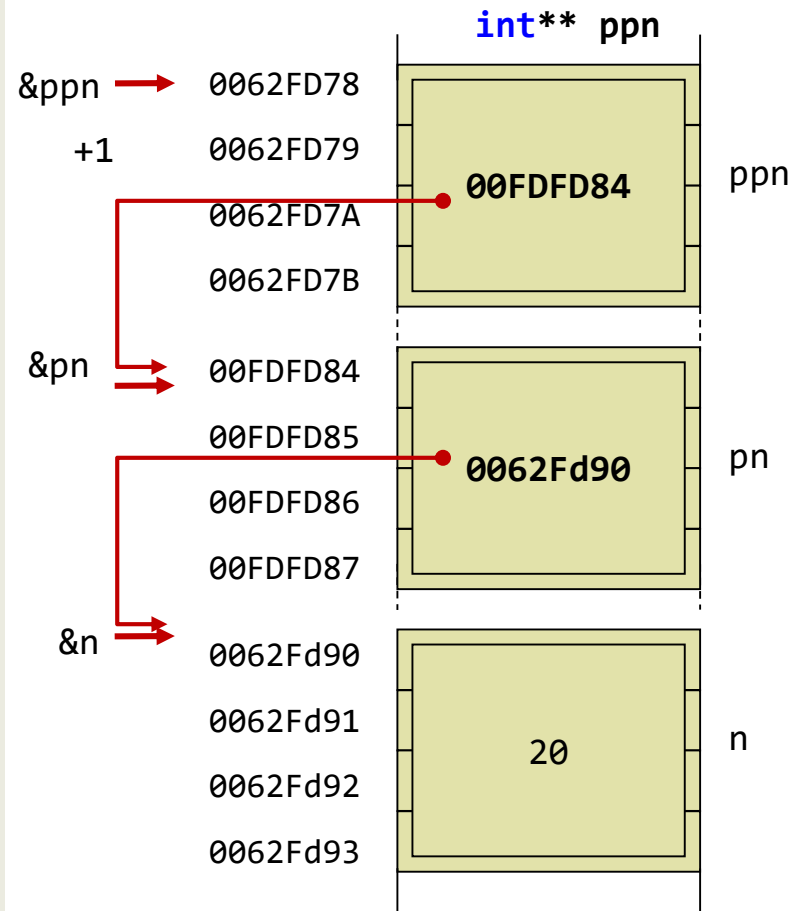
■ How does the int pointer work?

```
#include <stdio.h>

int main()
{
    int n = 10;
    int* pn = &n;
    int** ppn = &pn;

    printf("%p %p\n", pn, ppn);
    printf("%p %p\n", pn + 1, ppn + 1);
    printf("%p %p\n", &n, &n + 1);
    printf("%p %p\n", &pn, &ppn);
    printf("%p %p\n", &pn + 1, &ppn + 1);

    return 0;
}
```



Array and Pointer

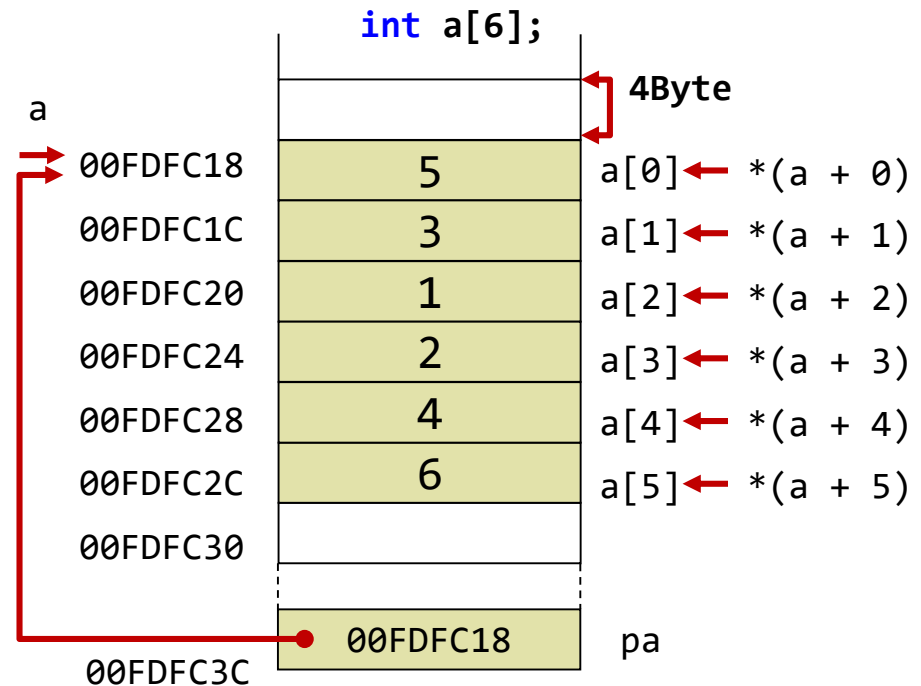
- The pointer to the first element of the array can be used as the name of the array.

```
#include <stdio.h>

int main()
{
    int a[6] = { 5, 3, 1, 2, 4, 6 };
    int* pa = a;

    printf("%d %d\n", *a, *pa);
    printf("%p %p\n", a, pa);
    printf("%p %p\n", &a, &pa);

    printf("%d %d\n", a[0], pa[0]);
    printf("%d %d\n", a[1], pa[1]);
    return 0;
}
```



Example: Array and Pointer

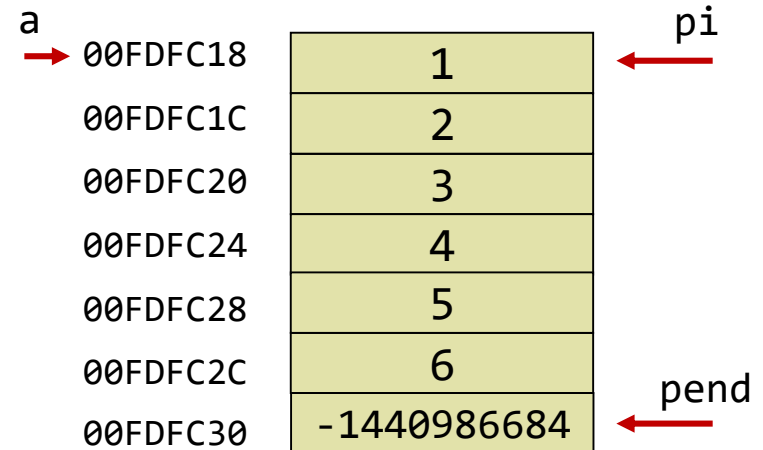
■ Is it a correct code?

```
#include <stdio.h>

int main()
{
    int a[6] = { 1, 2, 3, 4, 5, 6 };
    int *pend = a + 6;
    int *pi = NULL;

    for (pi = a; pi < pend; pi++)
        printf("%d\n", *pi);

    return 0;
}
```



Example: Array and Pointer

- Print the smallest array value using pointers.

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

```
int main()
```

```
{
```

```
    int a[6] = { 32, 12, 31, 42, 15, 24 };
```

```
    int *pend = a + 6;
```

```
    int *psmallest = a;
```

```
    int *pi = NULL;
```

```
    for (pi = a; pi < pend; pi++)
```

```
    {
```

```
        if (*pi < *psmallest)
```

```
            psmallest = pi;
```

```
    }
```

```
    printf("%d", *psmallest);
```

```
    return 0;
```

```
}
```

a

→ 00FDFC18
00FDFC1C
00FDFC20
00FDFC24
00FDFC28
00FDFC2C
00FDFC30

32
12
31
42
15
24
-1440986684

psmallest



pend



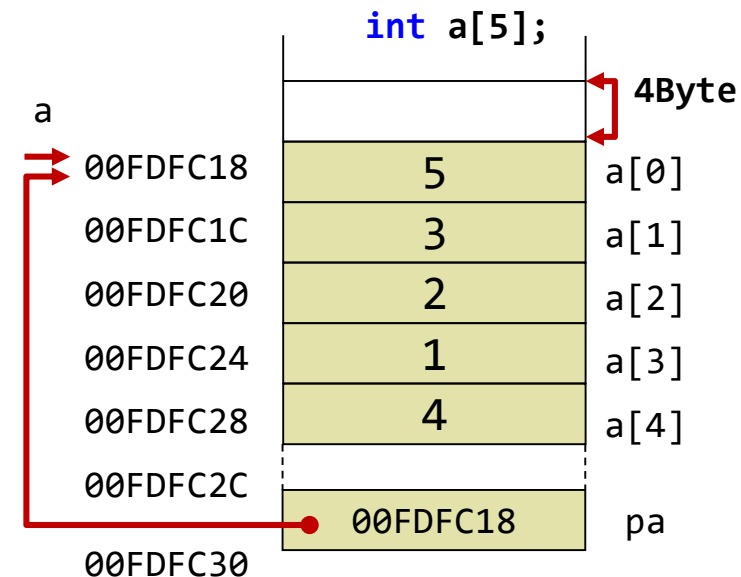
Passing Array to Function

- Passing a pointer (instead of an array) to a function

```
#include <stdio.h>
void printArray(int* pa, int len);

int main()
{
    int a[5] = { 5, 3, 2, 1, 4 };
    printArray(a, 5);
    return 0;
}

void printArray(int* pa, int len)
{
    int i;
    for (i = 0; i < len; i++)
        printf("%d\n", pa[i]);
}
```



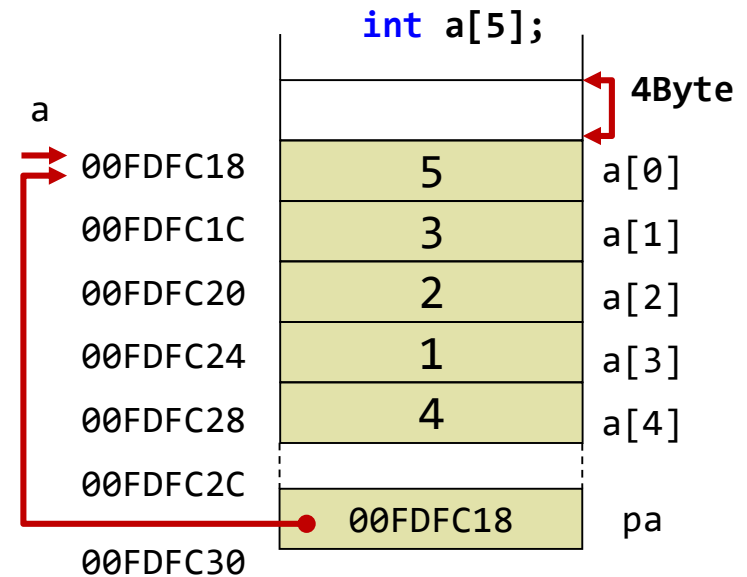
Example: Passing Array to Function

- Multiplying 4 for each element in an array

```
#include <stdio.h>
void multiply4(int* pa, int len);

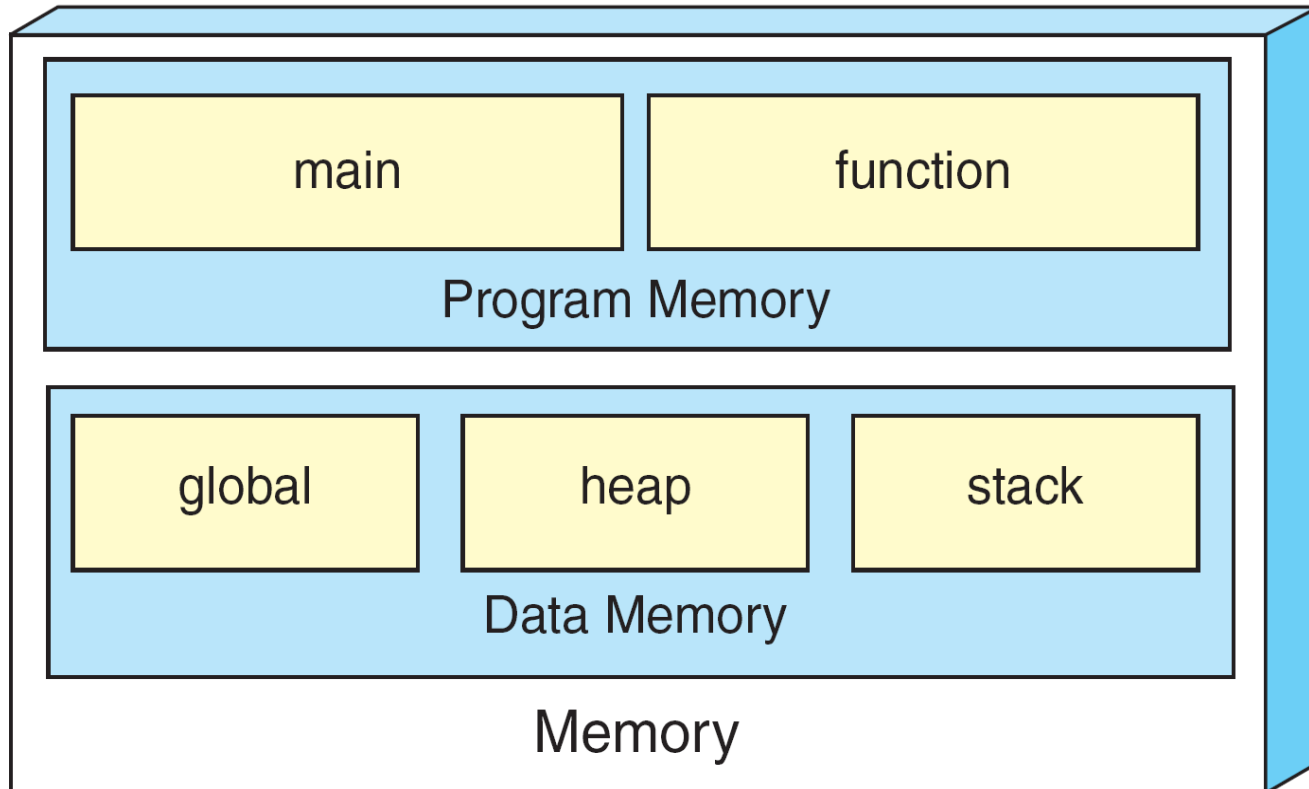
int main()
{
    int a[5] = { 5, 3, 2, 1, 4 }, i;
    multiply4(a, 5);
    for (i = 0; i < 5; i++)
        printf("%d\n", a[i]);
    return 0;
}

void multiply4(int* pa, int len)
{
    int i;
    for (i = 0; i < len; i++)
        pa[i] = pa[i] * 4;
}
```



How to Use Memory in C

- Conceptual view of memory used in C program



Dynamic Memory Allocation

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

int main()
{
    int size, i;
    scanf("%d", &size);
    // Allocate dynamic memory
    int* pn = malloc(sizeof(int)* size);
    for (i = 0; i < size; i++)
        scanf("%d", &pn[i]);

    for (i = 0; i < size; i++)
        printf("%d\n", pn[i]);

    free(pn); // Release memory

    return 0;
}
```

Dynamic Memory Allocation

■ Is it a correct code?

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

int *genNumbers(int size);

int main()
{
    int size, i;
    scanf("%d", &size);

    int *pn = genNumbers(size);
    for (i = 0; i < size; i++)
        printf("%d\n", pn[i]);

    return 0;
}
```

```
int *genNumbers(int size)
{
    int i;
    int *pn = malloc(4 * size);
    for (i = 0; i < size; i++)
        scanf("%d", &pn[i]);

    return pn;
}
```

Memory Leak

- It occurs when a computer program incorrectly manages memory allocations.
- Memory that is **no longer used** is not **released yet**.



Solving Memory Leak

- Use a pair of allocation and deallocation together!

```
#include <stdio.h>
#include <stdlib.h>
void genNumbers(int* pn, int s);

int main()
{
    int size, i;
    scanf("%d", &size);
    int* pn = malloc(4 * size);

    genNumbers(pn, size);
    for (i = 0; i < size; i++)
        printf("%d\n", pn[i]);

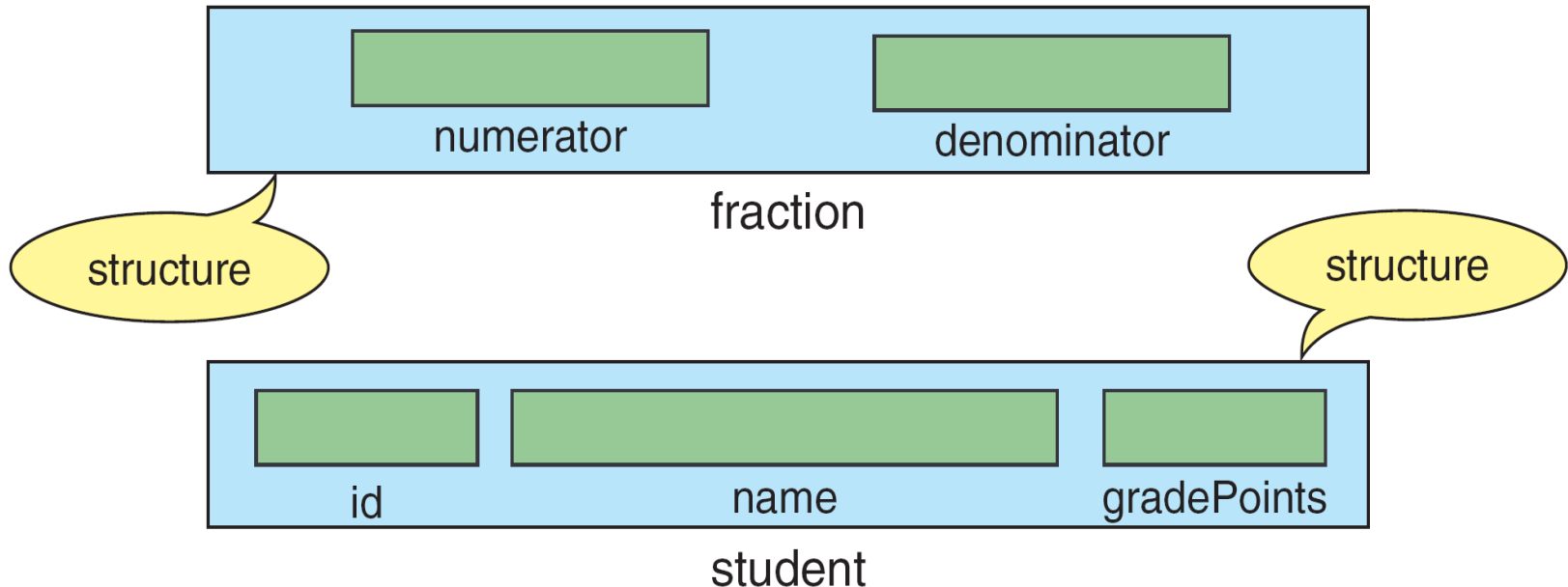
    free(pn);
    return 0;
}
```

```
void genNumbers(int* pn, int s)
{
    int i;
    for (i = 0; i < s; i++)
        scanf("%d", &pn[i]);
}
```

What is Structure?

■ Definition

- A collection of multiple related elements
- A single name including possibly several different types



What is Structure?

- How to declare a structure
 - All elements in the structure should be related **logically**.

```
#include <stdio.h>
typedef struct
{
    char name[10];
    int scores[3];
} STUDENT;

int main()
{
    STUDENT s1 = { "Alice", 80, 70, 60 };

    printf("%s\n", s1.name);
    for (int i = 0; i < 3; i++)
        printf("%d\n", s1.scores[i]);
    return 0;
}
```

-> (Arrow) Operator

- It is used to access the value of a structure pointer.

```
#include <stdio.h>
typedef struct
{
    char name[10];
    int scores[3];
} STUDENT;

int main()
{
    STUDENT s1 = { "Alice", 80, 70, 60 };
    STUDENT* s2 = &s1;

    printf("%s\n", s2->name);
    for (int i = 0; i < 3; i++)
        printf("%d\n", s2->scores[i]);
    return 0;
}
```

Example: Structure

■ Is it a correct code?

```
int main()
{
    STUDENT stu[1];
    for (int i = 0; i < 1; i++) {
        scanf("%s", stu[i].name);
        for (int j = 0; j < 3; j++) {
            scanf("%d", &stu[i].scores[j]);
            stu[i].total += stu[i].scores[j];
        }
    }
    for (int i = 0; i < 1; i++) {
        printf("%s\n", stu[i].name);
        for (int j = 0; j < 3; j++)
            printf("%d\n", stu[i].scores[j]);
        printf("%d\n", stu[i].total);
    }
    return 0;
}
```

```
typedef struct
{
    char name[10];
    int scores[3];
    int total;
} STUDENT;
```


Example: Structure

■ Structure with dynamic memory allocation

```
int main()
{
    int n;
    scanf("%d", &n);

    STUDENT* s = malloc(sizeof(STUDENT)*n);
    for (int i = 0; i < n; i++) {
        scanf("%s", s[i].name);
        s[i].total = 0;
        for (int j = 0; j < 3; j++) {
            scanf("%d", &s[i].scores[j]);
            s[i].total += s[i].scores[j];
        }
    }
    free(s);
    return 0;
}
```

```
typedef struct
{
    char name[10];
    int scores[3];
    int total;
} STUDENT;
```

Example: Structure

■ Multiplying two fractions

```
#include <stdio.h>

int main()
{
    FRACTION f1 = { 4, 5 };
    FRACTION f2 = { 3, 7 };
    FRACTION f3;

    f3.numerator = f1.numerator * f2.numerator;
    f3.denominator = f1.denominator * f2.denominator;

    printf("%d / %d", f3.numerator, f3.denominator);
    return 0;
}
```

```
typedef struct
{
    int numerator;
    int denominator;
} FRACTION;
```

Example: Structure

■ Summing two fractions

```
#include <stdio.h>

int main()
{
    FRACTION f1 = { 4, 5 };
    FRACTION f2 = { 3, 7 };
    FRACTION f3;

    f3.numerator = f1.numerator * f2.denominator;
    f3.numerator += f2.numerator * f1.denominator;
    f3.denominator = f1.denominator * f2.denominator;

    printf("%d / %d", f3.numerator, f3.denominator);
    return 0;
}
```

```
typedef struct
{
    int numerator;
    int denominator;
} FRACTION;
```

How to Be a Good Programmer

- Tips for programming

- <https://www.oreilly.com/ideas/7-ways-to-be-a-better-programmer-in-2014>

- Websites for Coding Exercises

- Euler project: <http://euler.synap.co.kr/>
- Backjoon online judge: <https://www.acmicpc.net/>
- Algospot: <https://www.algospot.com/judge/problem/list/>