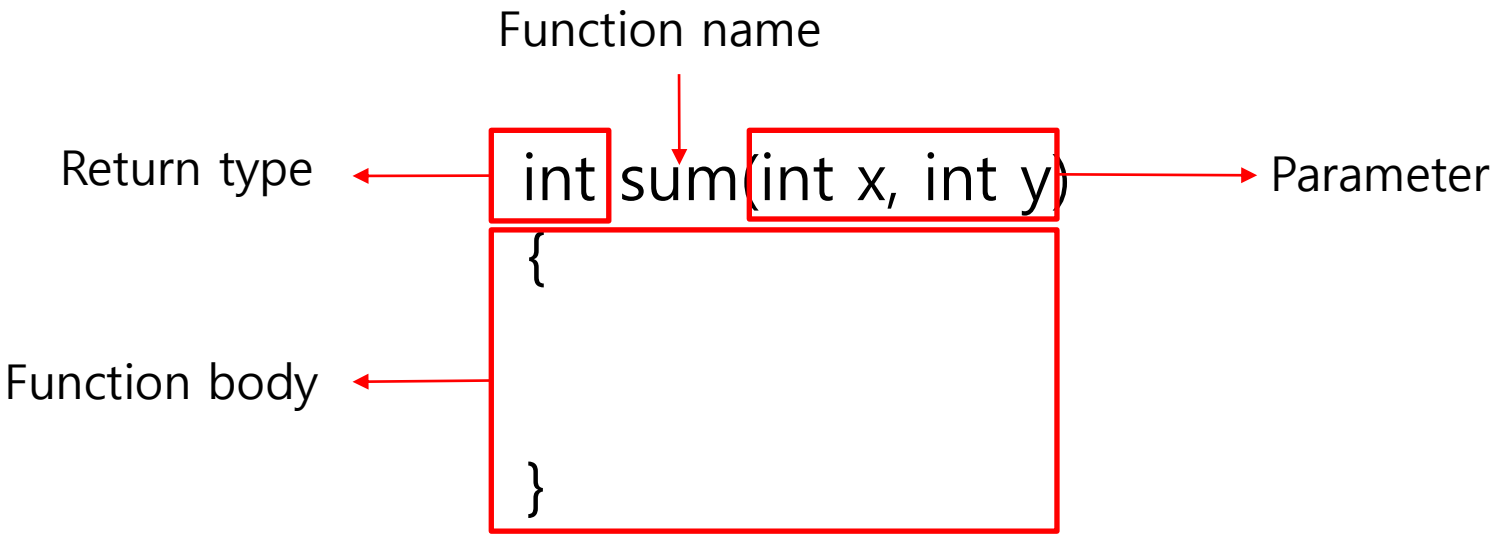
C언어 - HW3

임베디드스쿨1기
Lv1과정
2020. 08. 03
손표훈

1. 함수

1) 함수란? 하나의 특별한 목적의 작업을 수행하기 위해 독립적으로 설계된 프로그램 코드의 집합

2) 함수의 형태



(3) 함수의 장점

- 반복적인 코드 사용을 줄여 가독성을 향상시킴
- 모듈화를 통해 생산성을 향상시킴

1. 함수

Ex1) SUM FUNCTION

```c
#include <stdio.h>

int a=20;
int sum(int a, int b);

int main(void)
{
        int a= 10;
        int b = 20;
        int c = 0;

        printf("Value of a in main() = %d\n", a);
        c = sum(a,b);

        printf("Value of c in main() = %d\n", c);

        return 0;
}

int sum(int a, int b)
{
        printf("Value of a in sum() = %d\n", a);
        printf("Value of b in sum() = %d\n", b);

        return a+b;
}
```

```
Value of a in main() = 10
Value of a in sum() = 10
Value of b in sum() = 20
Value of c in main() = 30
```

-> a = 20이 아닌 이유 : C언어는 순차언어로 전역변수로 a = 20이 먼저 선언되었지만 사용되지 않았고, 다음 main문에서 a = 10으로 선언되어 함수에 사용된다.

1. 함수

Ex2) MAX FUNCTION

```c
#include <stdio.h>

int max(int num1, int num2);

int main(void)
{
        int a = 100;
        int b = 200;
        int ret;

        ret = max(a,b);

        printf("Max value is : %d\n", ret);

        return 0;
}

int max(int num1, int num2)
{
        int result;

        if(num1 > num2)
                result = num1;
        else
                result = num2;

        return result;
}
```

```
Max value is : 200
```

1. 함수

Ex2) SWAP FUNCTION

```c
#include <stdio.h>

void swap(int x, int y);

void swap(int x, int y)
{
        int temp;

        temp = x;
        x = y;
        y = temp;

        return;
}
int main(void)
{
        int a = 100;
        int b = 200;

        printf("Before swap, Value of a = %d\n", a);
        printf("Before swap, Value of b = %d\n", b);

        swap(a, b);
        printf("After swap, Value of a = %d\n", a);
        printf("After swap, Value of b = %d\n", b);

        return 0;
}
```

```
Before swap, Value of a = 100
Before swap, Value of b = 200
After swap, Value of a = 100
After swap, Value of b = 200
```

-> SWAP 함수 호출 시 parameter에 전달된건 100, 200이라는 “값(DATA)”만 함수에 전달되었다(x = 100, y = 200)

-> 함수내에서 DATA끼리만 SWAP을 진행하므로 함수 종료 후 main문안에서 a = 100, b = 200으로 동일하기 때문에 결과도 변화가 없다.

1. 함수

Ex2) SWAP FUNCTION2

```c
#include <stdio.h>

void swap(int *x, int *y);

void swap(int *x, int *y)
{
        int temp;

        temp = *x;
        *x = *y;
        *y = temp;

        return;
}
int main(void)
{
        int a = 100;
        int b = 200;

        printf("Before swap, Value of a = %d\n", a);
        printf("Before swap, Value of b = %d\n", b);

        swap(&a, &b);

        printf("After swap, Value of a = %d\n", a);
        printf("After swap, Value of b = %d\n", b);

        return 0;
}
```

```
Before swap, Value of a = 100
Before swap, Value of b = 200
After swap, Value of a = 200
After swap, Value of b = 100
```

-> “*”(포인터)를 paramete로 사용, “&”를 사용하여 a, b의 “주소 값”을 전달
-> *x = a의 “주소 값”, *y = b의 “주소 값“
-> 주소 값에 저장된 데이터를 SWAP함으로 결과적으로 a, b의 값이 서로 바뀐다.

2. 배열

1) 배열이란? 한가지 자료형을 연속적으로 나열한 것
2) 배열 선언

   ex) int Arr[10] -> Arr[0], Arr[1], Arr[2], Arr[3] ..... Arr[9]
   선언시 크기는 10으로 정하고, 사용시 요소값은 0부터 시작한다.
3) 배열 초기화

   ex) int Arr[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 20};
   int Arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 20};
   -> 배열의 크기를 정하지 않을 땐 반드시 초기화를 해줘야 한다.
   -> 배열 선언 시 컴파일러가 메모리 할당을 해줘야 하는데 크기를 정하지 않으면 컴파일시 에러가 발생한다.
   -> 초기화를 해주면 배열의 크기가 정해진다.
4) 다차원 배열

   ex) int Arr[3][4] -> 해석 시 4개짜리 배열이 3개 있는 것으로 해석

(5) 다차원 배열 초기화

ex) int Arr[3][4] = {{0, 1, 2, 3}, {4, 5, 6, 7}, {8, 9, 10, 11}};
int Arr[][4] = {{0, 1, 2, 3}, {4, 5, 6, 7}, {8, 9, 10, 11}};

2. 배열

Ex1) Array1

```c
#include <stdio.h>

int main(void)
{
        int n[10];
        int i,j;

        for(i = 0; i<10; i++)
        {
                n[i] = i + 100;
        }

        for(j = 0; j<10; j++)
        {
                printf("Element[%d] = %d\n", j, n[j]);
        }


        return 0;
}
```

```
Element[0] = 100
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
Element[7] = 107
Element[8] = 108
Element[9] = 109
```

2. 배열

Ex2) Array2

```c
#include <stdio.h>

int main(void)
{
        int a[5][2] = {{0,0}, {1,2}, {2,4}, {3,6}, {4,8}};
        int i,j;

        for(i = 0; i<5; i++)
        {
                for(j = 0; j<2; j++)
                {
                        printf("a[%d][%d] = %d\n", i,j,a[i][j]);
                }
        }
        return 0;
}
```

```
a[0][0] = 0
a[0][1] = 0
a[1][0] = 1
a[1][1] = 2
a[2][0] = 2
a[2][1] = 4
a[3][0] = 3
a[3][1] = 6
a[4][0] = 4
a[4][1] = 8
```

2. 배열

Ex3) getAverage

```c
#include <stdio.h>

double getAverage(int arr[], int size);
double getAverage(int arr[], int size)
{
        int i;
        double avg;
        double sum = 0;

        for(i = 0; i<size; ++i)
        {
                sum +=arr[i];
        }

        avg = sum/size;
        return avg;
}

int main(void)
{
        int balance[5] = {1000, 2, 3, 17, 50};
        double avg;

        avg = getAverage(balance, 5);

        printf("Average value is : %f\n", avg);
        return 0;
}
```

```
Average value is : 214.400000
```

3. 포인터

(1) 포인터란? 포인터는 메모리 “주소 값”을 저장하는 “변수”

(2) 포인터 변수선언

ex) int *ptr -> “*” asterisk로 포인터 변수 또는 주소에 저장된 데이터를 명시

ptr = &var -> “&” ampersand로 해당 변수가 가리키는 메모리 주소 값을 명시

(3) 포인터의 크기 : 시스템마다 다르다 32bit 시스템에선 4byte, 64bit에선 8byte의 크기를 가짐

*주의 : 포인터 선언 int는 저장된 데이터의 자료형의 크기이지 포인터의 크기가 아님!!

```c
#include <stdio.h>

int main(void)
{
        int var1;
        char var2[10];

        printf("Address of var1 variable : %x\n", &var1);
        printf("Address of var2 variable : %x\n", &var2);
        return 0;
}
```

```
Address of var1 variable : 12947ae8
Address of var2 variable : 12947aee
```

```c
#include <stdio.h>

int main(void)
{
        int var = 20;
        int *ip;

        ip = &var;

        printf("Address of var variable : %x\n", &var);

        printf("Address stored in ip variable : %x\n", ip);

        printf("value of *ip variable : %d\n", *ip);
        return 0;
}
```

```
Address of var variable : 129af63c
Address stored in ip variable : 129af63c
value of *ip variable : 20
```

3. 포인터

Ex1) Incrementing Pointer1

```c
#include <stdio.h>

const int MAX = 3;

int main(void)
{
        int var[] = {10, 100, 200};
        int i, *ptr;

        ptr = var;

        for(i = 0; i < MAX; i++)
        {
                printf("Address of var[%d] = %x\n", i, ptr);
                printf("Value of var[%d] = %d\n", i, *ptr);

                ptr++;
        }
        return 0;
}
```

```
Address of var[0] = da9aac6c
Value of var[0] = 10
Address of var[1] = da9aac70
Value of var[1] = 100
Address of var[2] = da9aac74
Value of var[2] = 200
```

-> ptr변수에 배열의 "이름"을 대입
-> 배열의 이름은 배열의 첫번째 요소의 주소를 명시한다!

3. 포인터

Ex2) Incrementing Pointer2

```c
#include <stdio.h>

const int MAX = 3;

int main(void)
{
        double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
        double *p;
        int i;

        p = balance;

        printf("Array values using pointer\n");


        for(i = 0; i < 5; i++)
        {
                printf("*(p+%d) : %f\n", i, *(p + i));
        }

        printf("Array values using balance as address\n");

        for(i = 0; i < 5; i++)
        {
                printf("*(balance + %d) : %f\n", i, *(balance+i));
        }

        return 0;
}
```

```
Array values using pointer
*(p+0) : 1000.000000
*(p+1) : 2.000000
*(p+2) : 3.400000
*(p+3) : 17.000000
*(p+4) : 50.000000
Array values using balance as address
*(balance + 0) : 1000.000000
*(balance + 1) : 2.000000
*(balance + 2) : 3.400000
*(balance + 3) : 17.000000
*(balance + 4) : 50.000000
```

-> 배열선언시 메모리는 순차적으로 할당됨을 알 수 있다!
-> 배열은 같은 자료형을 나열한것!

3. 포인터

Ex3) Return pointer from functions

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

int* getRandom(void)
{
        static int r[10];
        int i;

        srand((unsigned)time(NULL));

        for(i = 0; i < 10; ++i)
        {
                r[i] = rand();
                printf("r[%d] = %d\n", i, r[i]);
        }

        return r;
}
int main(void)
{
        int *p;
        int i;

        p = getRandom();

        for(i = 0; i < 10; i++)
        {
                printf("*(p + [%d]) : %d\n", i, *(p + i));
        }

        return 0;
}
```

```
r[0] = 1120131293
r[1] = 54110699
r[2] = 854533882
r[3] = 2140173913
r[4] = 863450559
r[5] = 29972119
r[6] = 52186606
r[7] = 520109716
r[8] = 409683081
r[9] = 1637846343
*(p + [0]) : 1120131293
*(p + [1]) : 54110699
*(p + [2]) : 854533882
*(p + [3]) : 2140173913
*(p + [4]) : 863450559
*(p + [5]) : 29972119
*(p + [6]) : 52186606
*(p + [7]) : 520109716
*(p + [8]) : 409683081
*(p + [9]) : 1637846343
```

-> 함수의 반환 자료형을 “자료형*”로 선언 해야함!

3. 포인터

Ex5) Passing pointers to function

```c
#include <stdio.h>

double getAverage(int *arr, int size);

int main(void)
{
        int balance[5] = {1000, 2, 3, 17, 50};
        double avg;

        avg = getAverage(balance, 5);

        printf("Average value is : %f\n", avg);

        return 0;
}

double getAverage(int *arr, int size)
{
        int i, sum = 0;
        double avg;

        for(i = 0; i < size; ++i)
        {
                sum += arr[i];
        }

        avg = (double)sum / size;

        return avg;
}
```

```
Average value is : 214.400000
```

-> 함수를 호출 시 parameter로 배열의 "이름"을 전달
-> 배열의 이름은 배열의 첫번째 요소의 주소를 명시한다!

3. 포인터

Ex6) pointer to pointer

```c
#include <stdio.h>

int main(void)
{
        int var;
        int *ptr;
        int **pptr;

        var = 3000;

        ptr = &var;

        pptr = &ptr;

        printf("Value of each variables\n");
        printf("Value of var = %d\n", var);
        printf("Value available at *ptr = %d\n", *ptr);
        printf("Value available at **pptr = %d\n", **pptr);

        printf("\nAddress of each variables\n");
        printf("Address of var = %x\n", &var);
        printf("Value of *ptr = %x\n", ptr);
        printf("Address of *ptr = %x\n", &ptr);
        printf("Value of **pptr = %x\n", pptr);

        return 0;
}
```

```
Value of each variables
Value of var = 3000
Value available at *ptr = 3000
Value available at **pptr = 3000

Address of each variables
Address of var = 6451e7e4
Value of *ptr = 6451e7e4
Address of *ptr = 6451e7e8
Value of **pptr = 6451e7e8
```

-> *ptr은 변수 var의 주소 값을 저장함!

-> **ptr은 변수 *ptr의 주소 값을 저장함!!!

3. 포인터

HW1) 메모리 완성- big endian?

```
int main(void)
{
        char c = 'a';
        int n = 7;
        double d = 3.14;
}
```

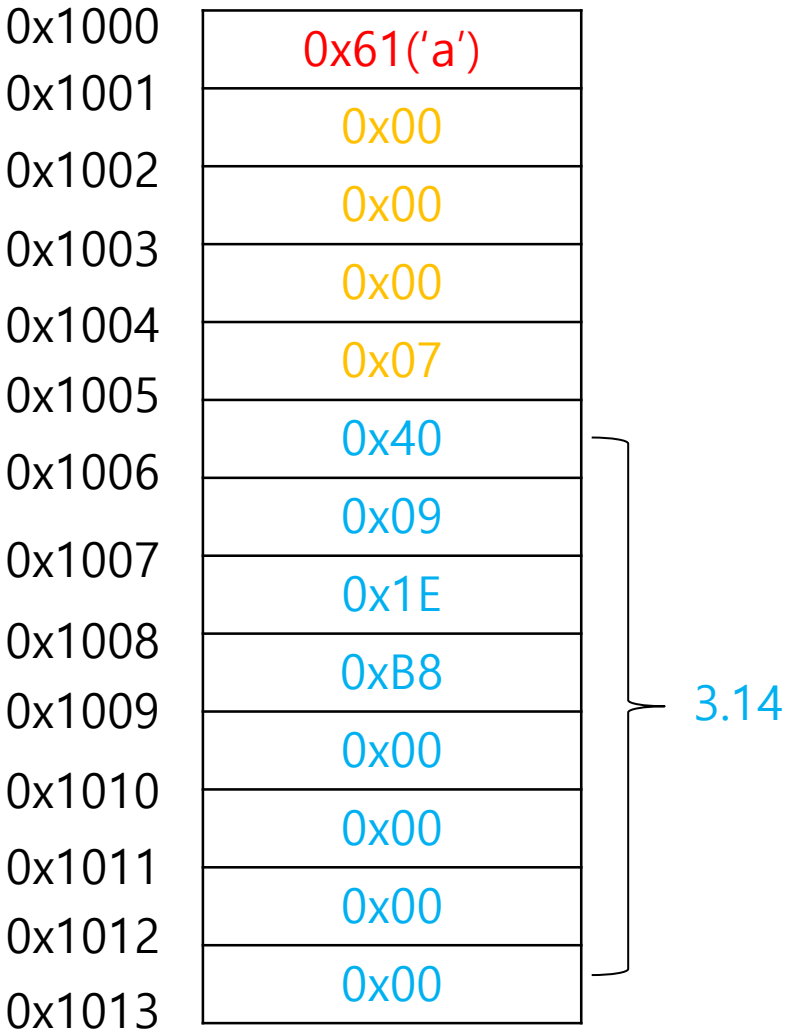


표1. int = 4byte

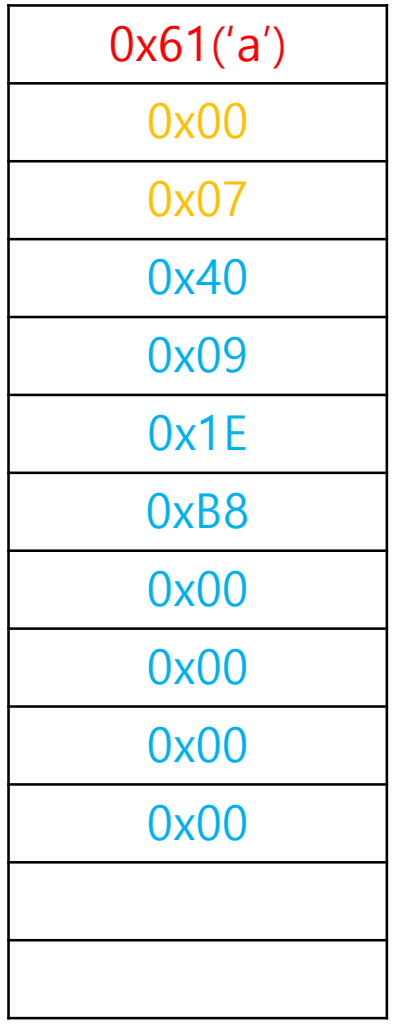


표2. int = 2byte

3. 포인터

※ Big endian? Little endian?

(1) MSB, LSB를 우선 알아보자!
Ex) 0x12345678이란 데이터가 있다면, (16진수로 '12'가 1byte임)
'12' -> MSB(Most Significant Byte) 최상위 바이트를 뜻한다.
'78' -> LSB(Least Significant Byte) 최하위 바이트를 뜻한다.

(2) Big endian이란? 데이터가 메모리에 저장될 때 MSB부터 저장됨

주소	0x1000	0x1001	0x1002	0x1003
데이터	0x12	0x34	0x56	0x78

(3) Little endian이란? 데이터가 메모리에 저장될 때 LSB부터 저장됨

주소	0x1000	0x1001	0x1002	0x1003
데이터	0x78	0x56	0x34	0x12

3. 포인터

HW2) 메모리 완성

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

* 3개짜리 배열이 2개 있음
* 배열의 이름은 배열의
  첫번째 요소의 주소를 명시

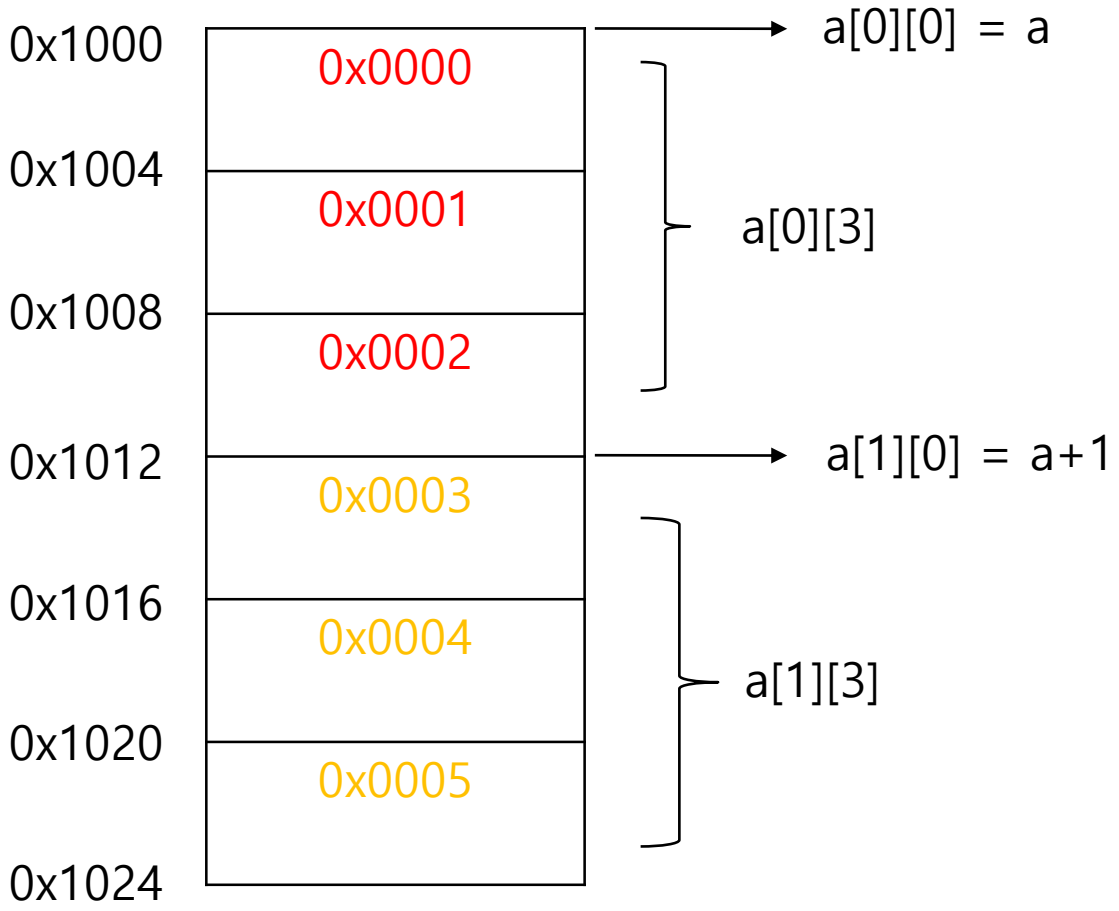


표1. int = 4byte

3. 포인터

HW2) 메모리 완성

1) &a = 0x1000
2) &a+1 = 없음
3) a+1 = 0x1012
4) *a = 0x1000
5) *a+1 = 0x1004
6) **a = 0x0000 = 0(a[0][0])

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

* 3개짜리 배열이 2개 있음
* 배열의 이름은 배열의
  첫번째 요소의 주소를 명시

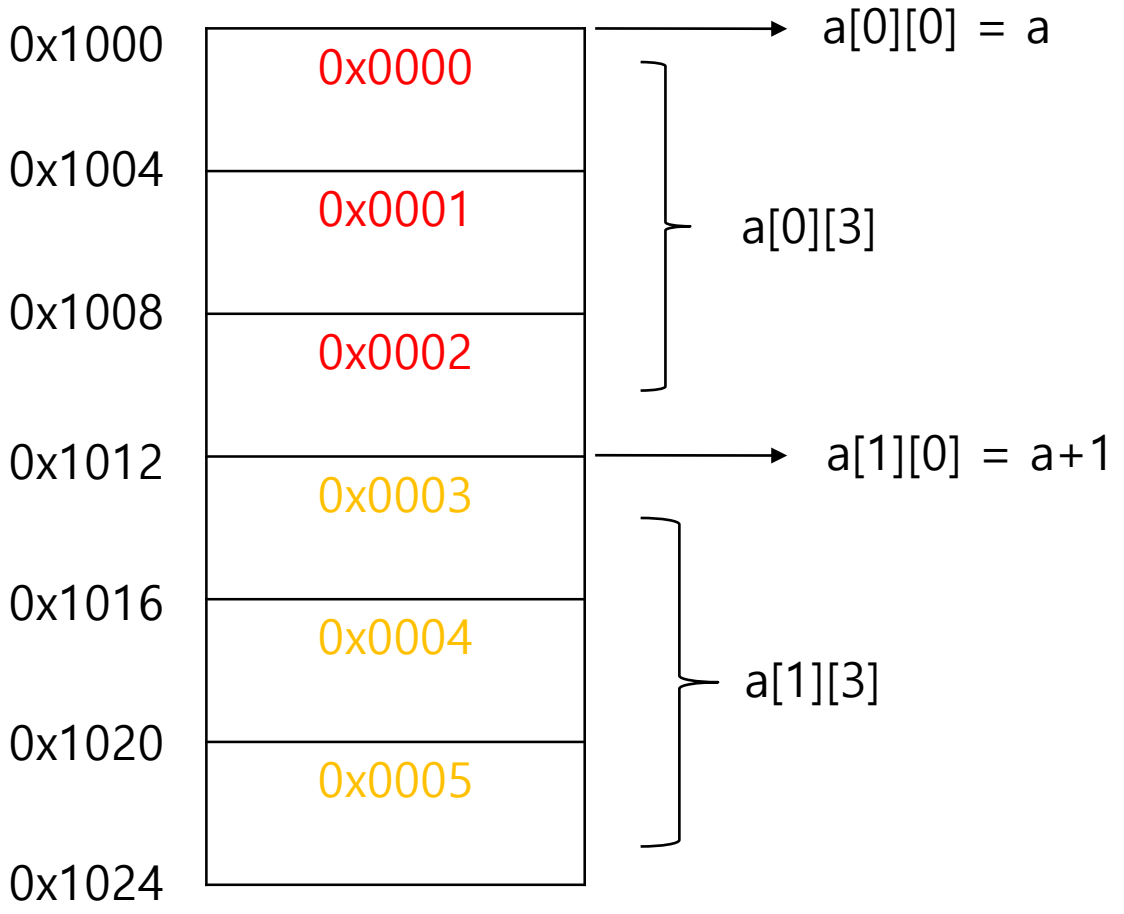


표1. int = 4byte

포기하면 얻는 건 아무것도 없다.

3. 포인터

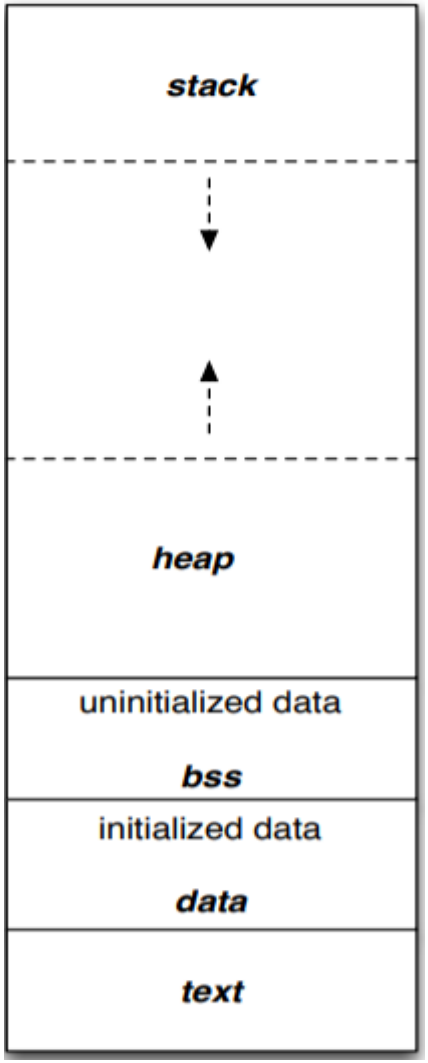
※ Data Segment?

-> 실제 작성된 코드는 메모리에 어떻게 할당이 될까?

1) STACK : 지역변수, 컴파일러에 의해 자동적으로 생성된 변수(&)가 저장되는 공간으로, LIFO구조(처음들어온 데이터가 먼저 나감) stack pointer라는 레지스터가 stack의 최상위 주소를 가지고 있으며, 데이터 입력 시 하위 주소쪽으로 값이 변경된다.

2) HEAP : malloc, calloc, realloc등과 같이 메모리 할당 함수에 의해 동적으로 메모리가 할당된 변수들이 저장된다.

3) 그 외 다음페이지 참조



3. 포인터

※ Data Segment?

-> 실제 작성된 코드는 메모리에 어떻게 할당이 될까?

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

int init = 0;
int global;
int sum(int a, int b);

int main(void)
{
    char string[] = "Hello World";
    char *ptr;
    static int num = 1;
    int num2 = 2;
    int num3 = 3;
    int result;

    ptr = (char *)malloc(sizeof(string));

    printf("%s\n", string);

    result = sum(num2, num3);
    printf("Sum = %d\n", result);
    return 0;
}

int sum(int a, int b)
{
    return a + b;
}
```

Data Segment	Data	Description
TEXT	실제 작성한 코드	#include <stdio.h> main문 Printf함수 sum함수
DATA	전역, static, const 초기화된 변수	int init static int num int num2 int num3
BSS	초기화 되지않은 static, 전역변수	int global
HEAP	동적 메모리영역	malloc함수로 메모리가 할당된 ptr변수
STACK	지역변수	int a,b,num2,num3,result char string[]