C언어 – HW2

임베디드스쿨2기

Lv1과정

2021. 03. 22

차현호

1. 변수

1) 변수란

변수는 특정한 데이터 타입을 저장할 수 있는 메모리 공간이며 다음과 같은 종류가 있다.

◆ 포인터 변수 : 특정한 타입의 메모리 주소를 저장할 수 있는 공간

아래의 코드를 보면 int형 포인터 변수의 값에 int형 데이터의 주소값을 넣는것을 확인 할 수 있다.

```c
#include <stdio.h>

int main(void)
{
    int data = 10;
    int* pData = &data;


    return 0;

}
```

1. 변수

작성한 코드를 gdb 디버거로 실행시켜 실제 pData 변수의 값과 data 변수의 주소값이 같은것을 확인 할 수 있다.

```
(gdb) p &data
$2 = (int *) 0x7fffffffdf3c
(gdb) p pData
$3 = (int *) 0x7fffffffdf3c
(gdb) p &pData
$4 = (int **) 0x7fffffffdf40
```

1. 변수

◆ 배열 변수 : 특정한 타입의 연속된 메모리 주소를 저장할 수 있는 공간

아래의 코드를 보면 크기가 10인 int형 타입의 배열을 선언하고 배열값을 for문을 이용해 10으로 초기화한 코드를 볼 수 있다.

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

int main(void)
{
    int i;
    int a[10] = {0,}; //initiallization 0


    //initiallization 10
    for (i = 0; i < 10; i++)
    {
        a[i] = 10;
    }

    return 0;
}
```

1. 변수

작성한 프로그램을 gdb로 실행하여 디버깅 해보면 배열의 첫번째 인자의 주솟값은 0x7fffffffdf20 이고 마지막 인자의 주솟값은 0x7fffffffdf48로 차이가 16진수로는 0x28 십진수로는 40바이트가 차이나는것을 확인 할 수 있다. 이로인해 4바이트 크기의 int형 변수 10개가 연속적으로 메모리에 저장되어있는 것을 확인 할 수 있었다.

```
(gdb) p &a[0]
$3 = (int *) 0x7fffffffdf20
(gdb) p &a[10]
$4 = (int *) 0x7fffffffdf48
(gdb) clear
Deleted breakpoint 1
```

1. 변수

◆ 함수 포인터 변수 : 특정한 프로토타입의 함수 주소를 저장할 수 있는 공간

아래의 코드를 보면 크기가 평소처럼 작성한 main 안에 변수가 선언되어있는것을 알수있다.

```c
#include <stdio.h>


int main(void)
{
    int data = 10;

    return 0;
}
```

1. 변수

마찬가지로 gdb디버거를 사용하여 main 함수의 주솟값을 출력해보면 0x555555554508 이라는것을 확인할 수 있다.

```
(gdb) p main
$1 = {int (void)} 0x555555554608 <main>
```

여기서 2가지 의문점이 발생하는데

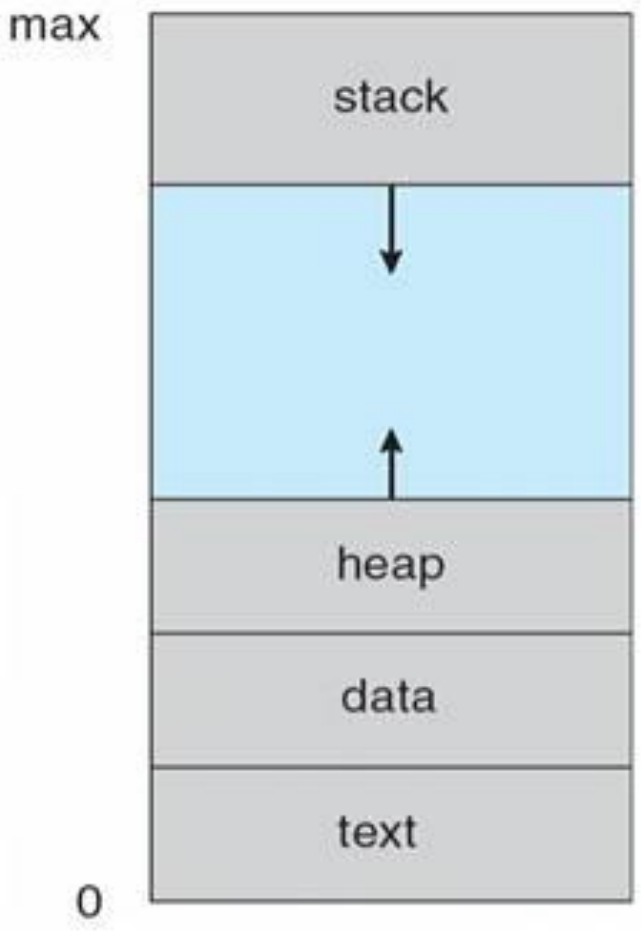
첫번째 의문점은 현재 사용하고 있는 컴퓨터의 메모리는 16GB 메모리를 사용하고 있으나 앞에서 확인했을 때의 주소값들은 전부 16GB 이상의 값인것을 확인 할 수 있었다 어떻게 실제 컴퓨터에서 사용하고 있는 메모리보다 큰 주소의 메모리 주소를 사용할 수 있을까?

두번째 의문점은 앞에서 일반변수의 주솟값은 0x7fffffffdf20 이였고 함수의 주소값은 이보다 훨씬 작은 0x555555554508이였다.

먼저 두번째 의문점의 해결 실마리는 메모리 구조를 살펴보면 알 수 있었다.

2. 메모리 구조

메모리구조는 빌드가 완료된 실행파일이 메모리에 적재되는 구조를 나타낸 것이며 아래 그림과 같다.



그림을 보면 낮은 주소부터 차례대로 text -> data -> heap -> stack 순으로 되어있는 것을 확인 할 수 있다.

이러한 영역들은 실제 프로그래밍을 하면서 어느부분에 해당하는지 확인해 보자

출처 : https://stackoverflow.com/

2. 메모리 구조

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

int g_number1; // data area(bss)
int g_number2 = 10; // data area(data)


int main(void) // text area
{

    int number1 = 10; // stack area
    int* number2;
    static int number3; // data area

    number2 = (int*) malloc(sizeof(int)); // heap area


    free(number2);

    return 0;
}
```

이전 슬라이드의 메모리 구조를 기억하면서 코드를 봐보면 전역변수 및 static 변수들은 data 영역에 저장되고 우리가 지금까지 main 문안에서 선언했던 변수들은 stack 영역에 저장되는것을 볼 수 있다.

또한 main 함수는 text영역에 저장 되는것을 볼 수 있다.

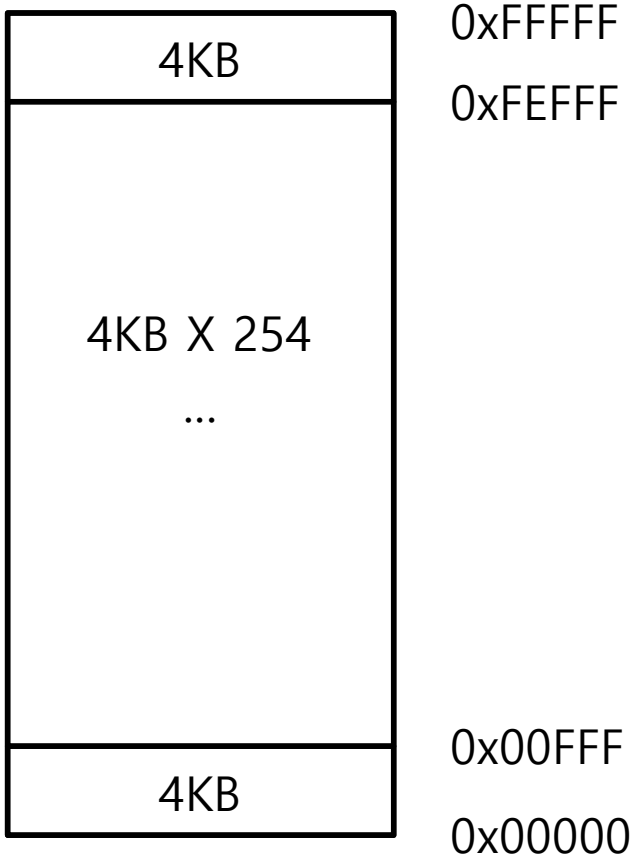
이렇게 두번째 의문점이였던 함수 주소값과 변수의 주소값이 왜이렇게 차이나는가는 해결이 되었다.

다음으로 첫번째 의문점을 해결해보자.

3. 메모리 관리기법

앞에서 우리는 변수의 주소가 실제 PC 물리 메모리 주소보다 큰 숫자인것을 확인 할 수 있었는데 이것이 가능한이유를 알아보기위해 먼저 메모리 관리 기법에 대해 알아보자

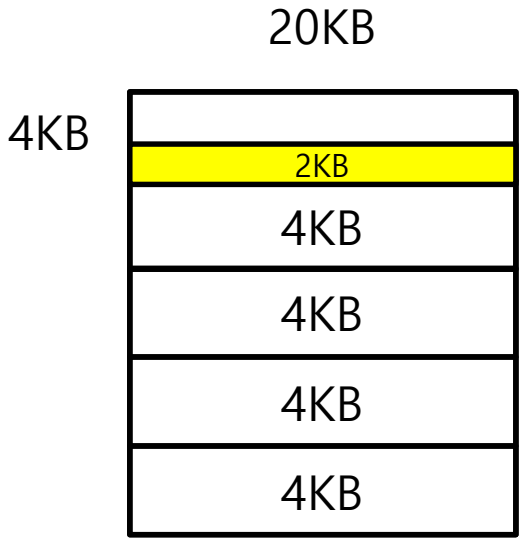
우리가 사용하는 물리메모리의 최소단위는 페이지 프레임이라고 부르며 크기는 4KB를 기본단위로 사용한다.



크기가 1MB 인 메모리를 사용한다면 옆의 그림과 같이
4KB크기의 페이지 프레임을 256개 개 가질 수 있다.
4KB X 256 = 1MB

3. 메모리 관리기법

이렇게 페이지 프레임 단위로 메모리를 할당하면 메모리 단편화 문제가 발생하는데 메모리 단편화 문제는 외부 메모리 단편화와 내부 메모리 단편화 2가지가 존재한다.
먼저 내부 메모리 단편화를 살펴보면

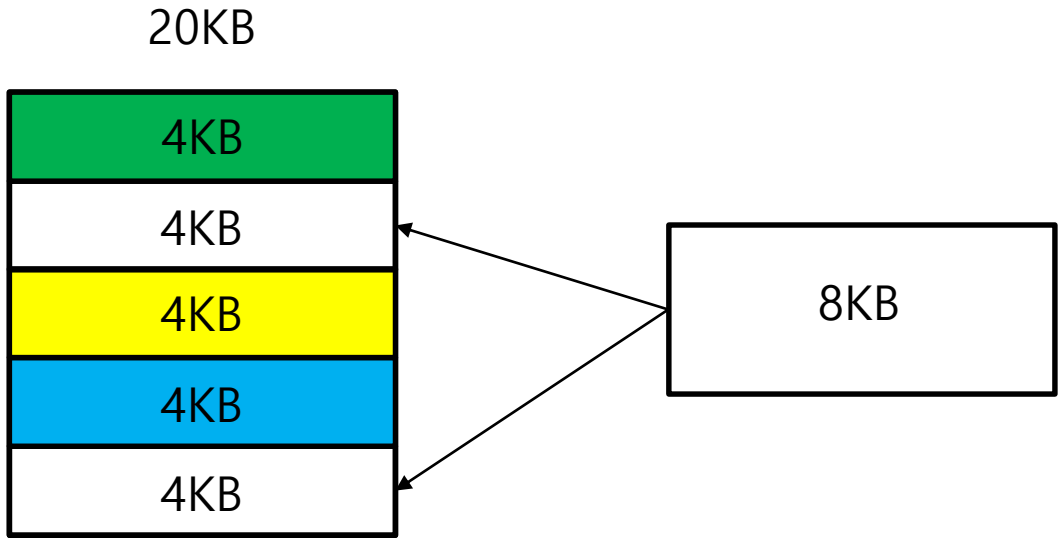


내부 메모리 단편화

왼쪽 그림을 보면 페이지 프레임 단위보다 작은 2KB의 메모리만 사용하였을때 2KB가 낭비되는것을 확인 할 수 있다.

3. 메모리 관리기법

다음으로 외부 단편화를 살펴보자 아래 그림을 보면 색칠해져있는부분은 이미 메모리가 할당된 영역이다. 이때 새롭게 8KB 할당요청이 들어올때 4KB가 서로 파편으로 나뉘어져 있어 8KB 메모리 할당이 불가능해지는것을 외부 단편화라고 한다.



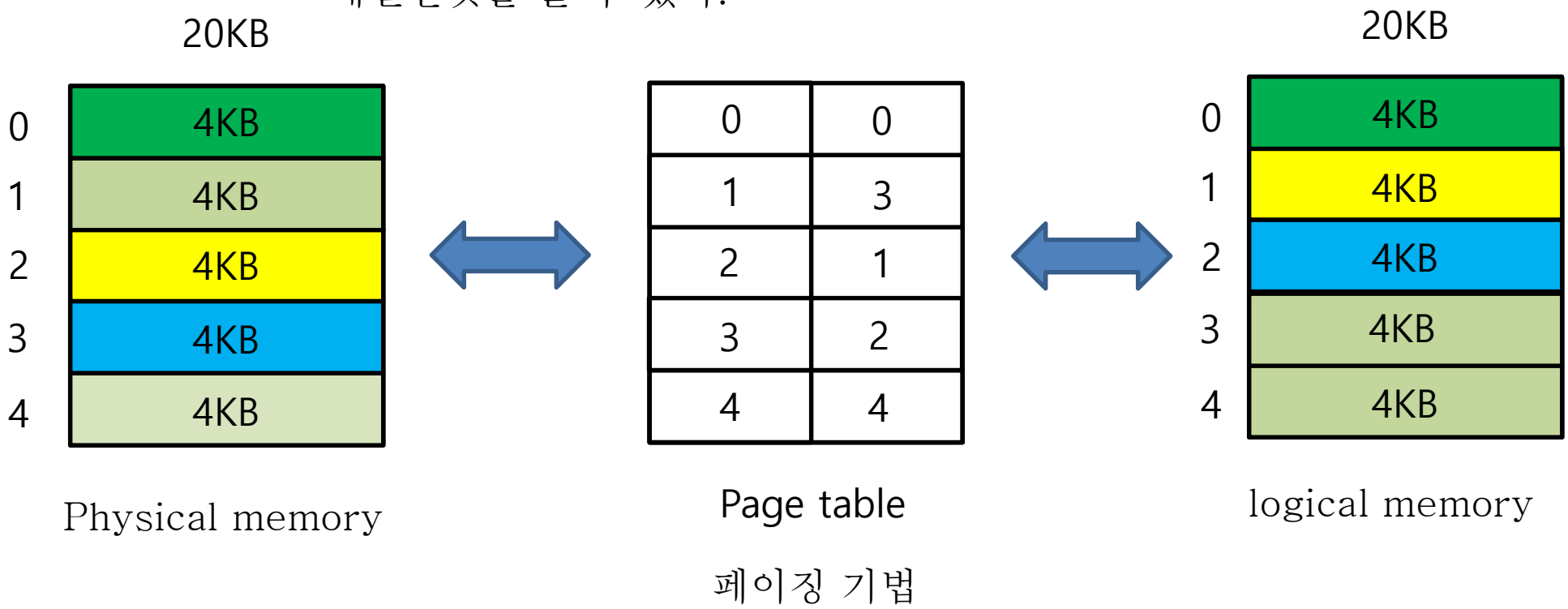
외부 메모리 단편화

3. 메모리 관리기법

이러한 메모리 단편화 문제를 해결하기 위한 방법으로 페이징 기법과 세그먼테이션 기법이 있다.

페이징 기법 -> 외부 단편화 해결
세그멘테이션 기법 -> 내부 단편화 해결

먼저 페이징 기법을 살펴보면 앞에서 8KB 메모리할당을 문제를
페이지 테이블을 이용하여 마치 연속된 메모리처럼 보이게 해주어
해결한것을 볼 수 있다.



페이징 기법

3. 메모리 관리기법

페이징기법을 보면 처음에 가졌던 의문점을 해결할수있다. 정리하자면 우리가 보았던 변수의 주소들은 가상의 주소이고 실제 할당된 물리 메모리는 메모리 관리 기법등을 통해 16GB안에 할당된 것을 알 수 있다.

4. 문제 풀이

1. 0 ~ 100 까지 숫자 중 홀수만 출력하시오

```
int printOddNumber(int range)
{
    int i;
    int numberCount = 0;

    if (range < 0)
    {
        return 0;
    }

    for (i = 0; i < range; i++)
    {
        if (i % 2 == 1)
        {
            printf("%2d ", i);
            numberCount++;
            if (numberCount % 10 == 0)
            {
                printf("\n");
            }
        }
    }

    return 1;
}
```

풀이 : 입력된 범위 (문제에서는 0~100) 동안 for 문을 반복하면서 2로 나누었을때 나머지가 1인 숫자들을 프린트한다. 이때 10개씩 줄바꿈을 해야하므로 숫자 카운트가 10개일때마다 줄바꿈 코드를 넣어주었다.

```
 1  3  5  7  9 11 13 15 17 19
21 23 25 27 29 31 33 35 37 39
41 43 45 47 49 51 53 55 57 59
61 63 65 67 69 71 73 75 77 79
81 83 85 87 89 91 93 95 97 99
```

4. 문제 풀이

2. 1 ~ 50 까지 숫자의 합을 구하는 프로그램을 작성하세요.

```
int continousSumLoop(const int range)
{
    int result = 0;
    int i;

    if (range <= 0)
    {
        return 0;
    }


    for (i = 1; i <= range; i++)
    {
        result += i;
    }

    return result;
}
```

풀이 : 입력된 범위 (문제에서는 1~50) 동안 for 문을 반복하면서 1부터 50까지 더하여 1275라는 값을 얻을 수 있었다.

```
int main()
{

    assert(problem1() == 1);

    assert(problem2() == 1275);
    assert(problem2_loop() == 1275);

    assert(problem3() == 198);
    assert(problem3_loop() == 198);

    assert(problem4() == 4233);
    assert(problem4_loop() == 4233);

    assert(problem5() == 55);

    assert(problem6() == 5);
    printf("Perfect !! \n");

    return 0;
}
```

4. 문제 풀이

2. 1 ~ 50 까지 숫자의 합을 구하는 프로그램을 작성하세요.

```
int continousSum(const int range)
{
    int result = 0;

    if (range <= 0)
    {
        return 0;
    }

    result = (range * (2 + (range - 1))) / 2;

    return result;
}
```

풀이 :
for문을 이용한 풀이는 시간복잡도가 O(n)이 걸린다.
따라서 더하는 숫자가 커질수록 연산에 걸리는 시간이 오래걸리는 것인데 이를 O(1)로 줄이기 위해 초항 1, 공차 1 인 등차수열의 합 공식을 이용하여 코드작성을 하였다.

$$S_n = \frac{n\{2a+(n-1)d\}}{2}$$

출처 :
https://bhsmath.tistory.com/17

4. 문제 풀이

3. 1 ~ 33 까지 3의 배수의 합만 구해보세요.

```
int multipleSumLoop(const int range, const int multiple)
{
    int result = 0;
    int i;

    if (multiple <= 0 || range <= 0)
    {
        return 0;
    }

    for (int i = 1; i <= range; i++)
    {
        if (i % multiple == 0)
        {
            result += i;
        }
    }


    return result;

}
```

풀이 :
정해진 범위 (1~ 33)까지 for 문을 돌면서 3으로 나누었을때 나머지가 0인 값들을 전부 더하였다.

4. 문제 풀이

3. 1 ~ 33 까지 3의 배수의 합만 구해보세요.

```
int multipleSum(const int range, const int multiple)
{
    int divisionQuotient = 0;
    int result = 0;

    if (multiple <= 0 || range <= 0)
    {
        return 0;
    }

    divisionQuotient = range / multiple;

    result = (divisionQuotient * (2 * multiple + (divisionQuotient - 1) * multiple)) / 2;

    return result;
}
```

풀이 :
2번 문제와 마찬가지로 등차수열의 합 공식을 사용하여 시간복잡도를 O(1)로 줄였다.

4. 문제 풀이

4. 1 ~ 100의 숫자 중 2의 배수의 합과 3의 배수의 합을 각각 구해봅시다.

```c
int problem4_loop(void)
{
    int result1 = 0;
    int result2 = 0;

    result1 = multipleSumLoop(100, 2);
    result2 = multipleSumLoop(100, 3);

    printf("1 ~ 100 까지 2의 배수의 합 : %d \n", result1);
    printf("1 ~ 100 까지 3의 배수의 합 : %d \n", result2);

    return 1;
}
```

```
1 ~ 100 까지 2의 배수의 합 : 2550
1 ~ 100 까지 3의 배수의 합 : 1683
```

풀이 :
앞의 3번문제에서 작성한 함수(loop)를 이용하여 2의 배수와 3의배수의 합을 각각 구했다..

4. 문제 풀이

4. 1 ~ 100의 숫자 중 2의 배수의 합과 3의 배수의 합을 각각 구해봅시다.

```c
int problem4(void)
{
    int result1 = 0;
    int result2 = 0;

    result1 = multipleSum(100, 2);
    result2 = multipleSum(100, 3);


    printf("1 ~ 100 까지 2의 배수의 합 : %d \n", result1);
    printf("1 ~ 100 까지 3의 배수의 합 : %d \n", result2);

    return 1;
}
```

```
1 ~ 100 까지 2의 배수의 합 : 2550
1 ~ 100 까지 3의 배수의 합 : 1683
```

풀이 :
앞의 3번문제에서 작성한 함수(등차수열합 공식)를 이용하여 2의 배수와 3의배수의 합을 각각 구했다.

4. 문제 풀이

5. 피보나치 수열의 n 번째 항을 구하는 프로그램을 만들어봅시다.

```
int fibonacciNumbers(int index)
{
    int result = 0;
    int number1 = 1;
    int number2 = 1;
    int i;

    if (index <= 0)
    {
        return 0;
    }
    else if (index <= 2)
    {
        return 1;
    }

    for (i = 3; i <= index; i++)
    {
        result = number1 + number2;
        number1 = number2;
        number2 = result;
    }

    return result;
}
```

풀이 :
피보나치수열은 첫째항 1 둘째항 1 이며 셋째항부터 앞의 두개의 항을 더한 결과 값이다.

따라서 n번째항이 인덱스로 들어오면 for문에서 3번째항부터 반복문을 실행한다.이때 결과는 앞의 두개의 숫자를 더한 값이 되고 앞의 두개의 숫자는 매번 반복문이 실행될때마다 업데이트를 해준다.

답 : 1251

```
int problem6(void)
{
    int result = 0;

    result = sequenceNumber(25);

    return result;
}
```

4. 문제 풀이

6. 1, 1, 1, 1, 2, 3, 4, 5, 7, 10, 14, 19, 26, 36, 50, ... 으로 진행되는 수열이 있다.
여기서 25번째 항의 숫자값을 구해봅시다.

```
int sequenceNumber(int index)
{
    int result = 0;
    int number1 = 1;
    int number2 = 1;
    int number3 = 1;
    int number4 = 1;
4   int i;

    if (index <= 0)
    {
        return 0;
    }
    else if (index <= 4)
    {
        return 1;
    }

    for (i = 5; i <= index; i++)
    {
        result = number1 + number4;
        number1 = number2;
        number2 = number3;
        number3 = number4;
        number4 = result;
    }

    return result;
}
```

풀이 :
주어진 문제의 수열은 피보나치수열과 유사한 패턴을 가지고 있다.
피보나치 수열과 다른점은 n번째 항과 n+4번째항을 더한값이
다음항이라는 점이다. 풀이방법은 피보나치 수열과 유사한
방식으로 풀이하였다.

4. 테스트 코드

```c
int main()
{

    assert(problem1() == 1);

    assert(problem2() == 1275);
    assert(problem2_loop() == 1275);

    assert(problem3() == 198);
    assert(problem3_loop() == 198);

    assert(problem4() == 1);
    assert(problem4_loop() == 1);

    assert(problem5() == 55);

    assert(problem6() == 1251);

    printf("Perfect !! \n");

    return 0;
}
```

5. 질문

Q1. 앞서 메모리 구조를 보면 main의 함수포인터 주소값은 text 영역에 저장되고 static 변수는 data 영역에 저장되어야한다. 그러나 실제 코드를 작성해서 주소를 보면 main의 함수포인터 주소값과 static 변수의 주소값은 같은 영역에 있는것처럼 보인다. 이유가 궁금합니다.

Q2. 내부메모리 단편화 해결방법인 세그멘테이션 기법과 외부메모리 단편화 해결방법인 페이징 기법의 차이점을 알고싶습니다.

Q3. Data type관련해서 이전에 ATmega128 mcu 사용시에 int형 타입의 크기는 4바이트가 아닌 2바이트였는데 int는 반드시 4바이트가 아닐수가 있는지?