C언어 - HW3

임베디드스쿨2기
Lv1과정
2021. 04. 01
차현호

1. 비트 연산자

1) 비트연산자란
데이터의 최소 단위를 비트라고 하며 이 비트들을 이용한 AND, OR XOR들을 비트 연산자라고 한다.

1. AND 연산
서로 같은 자리수의 비트가 둘다 1(참)이며 1 그외는 전부 0인 연산자이며
c언어에서는 &로 표현되고 관계연산시에는 &&이 사용된다.
진리표는 다음과 같다.

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

1. 비트 연산자

2. OR 연산
서로 같은 자리수의 비트중 1개 만 1이되어도 참이 되며 c언어에서는 | 로 표현되며 관계 연산자는 ||이다.
진리표는 아래와 같다.

X	Y	XY
0	0	0
0	1	1
1	0	1
1	1	1

1. 비트 연산자

3. NOT 연산
비트값을 반전시킨다 1->0, 0->1 c프로그램에서는 ~로사용하고 관계연산자는 !로 사용된다.

X	~X
0	1
1	0

1. 비트 연산자

4. XOR 연산
서로 같은 자리수의 비트가 서로 다르면 참 같으면 거짓이 된다. C에서는 ^로 표시되고 관계연산자는 없다.
XOR 연산은 암호화 과정에서 많이 사용된다.

X	Y	XY
0	0	0
0	1	1
1	0	1
1	1	0

1. 비트 연산자

1) 2의 보수 쉽게 구하기
컴퓨터는 음수를 표현할때 2의 보수를 이용하여 표현한다. 2의보수의 변환방법은 다음과같다.

1. 뒤쪽에서 가장 먼저 나오는 1을 찾는다.
2. 맨 뒤에서 1까지의 숫자들은 그대로 유지한다.
3. 1 이후의 숫자들은 전부 반전을 시킨다.

예) 37은 이진수로 00100101 이다 뒤쪽에서 가장먼저나오는 1을 찾아
그전숫자들은 그대로 유지한채 반전시키면 11011011이되며 -37을 의미한다.

1. 비트 연산자

1) XOR 활용

대소문자 변환시 0x32와 XOR 연산을 함으로써 쉽게 활용이 가능하다.

```
  1000001 (65) A
^0100000 (32)
-------------------
  1100001 (97) a
```

1. 비트 연산자

1) AND NOT

AND NOT의 사용처는 2의 n승 단위 정렬을 하고자 하는 경우에 매우 유용하며 보통 리눅스 커널의 페이지 프레임의 페이지 크기를 정렬하는 목적으로 많이 사용한다.

예) 0~1000까지 64의 배수를 알고 싶으면 아래의식과 AND NOT을 하면된다.

식 : 2^n – 1 (n = 7)

풀이 : 2^7 – 1 = 0b111111 -> NOT 연산 -> 0b000000

```
  11 1110 1000 (1000)
&11 1100 0000 ~(2^7 - 1)
------------------------
   11 1100 0000 -> 64 + 128 +256 +512
```

1) 함수선언방법

C언어에서의 함수 선언 방법은 아래와 같다.

```
int function(int number) -> 함수 프로토 타입

int function(int number) -> 함수 정의
{
    return number + 1;
}
```

위그림과 같이 함수의 형태는 리턴타입 함수이름(매개변수) 이러한 형식으로 되어있다.

그렇다면 함수 이름이 같으면 어떻게 될까?

1) 함수오버로딩

전슬라이드의 함수와 똑같은 함수를 선언하면 어떻게 되는지 확인해보자.

```c
#include <stdio.h>

int function(int number);
int function(int number);

int main(void)
{
    int result = 0;

    result = function(1);

    printf("Result = %d \n", result);
    return 0;
}

int function(int number)
{
    return number + 1;
}

int function(int number)
{
    return number + 2;
}
```

똑같은 함수 2개를 선언하면 아래와 같은 에러가 발생한다.

그럼 c++에서 지원되는 함수오버로딩은 가능할까?

```
function_test.c:21:5: error: redefinition of ‘function’
 int function(int number)
     ^~~~~~~~
function_test.c:16:5: note: previous definition of ‘function’ was here
 int function(int number)
     ^~~~~~~~
```

1) 함수오버로딩

함수 오버로딩을 테스트를 위해 아래와 같이 코드를 작성하고 컴파일해본 결과
c에서는 함수 오버로딩이 지원이 안된다는것을 알수있다.
이걸로 미루어보아 c언어에서는 함수이름만으로 함수를 구분한다는것을 알수있다.

```c
#include <stdio.h>

int function(int number);
int function(int number1, int number2);

int main(void)
{
    int result = 0;

    result = function(1, 2);

    printf("Result = %d \n", result);
    return 0;
}

int function(int number)
{
    return number + 1;
}

int function(int number1, int number2)
{
    return number1 + number2;
}
```

```
function_test.c:4:5: error: conflicting types for ‘function’
 int function(int number1, int number2);
     ^~~~~~~~
function_test.c:3:5: note: previous declaration of ‘function’ was here
 int function(int number);
     ^~~~~~~~
function_test.c:21:5: error: conflicting types for ‘function’
 int function(int number1, int number2)
     ^~~~~~~~
function_test.c:16:5: note: previous definition of ‘function’ was here
 int function(int number)
     ^~~~~~~~
eyl@eyl-VirtualBox:~$ vim function_test.c
eyl@eyl-VirtualBox:~$ gcc function_test.c
function_test.c:4:5: error: conflicting types for ‘function’
 int function(int number1, int number2);
     ^~~~~~~~
function_test.c:3:5: note: previous declaration of ‘function’ was here
 int function(int number);
     ^~~~~~~~
function_test.c:21:5: error: conflicting types for ‘function’
 int function(int number1, int number2)
     ^~~~~~~~
function_test.c:16:5: note: previous definition of ‘function’ was here
 int function(int number)
     ^~~~~~~~
```

3. 문제풀이

1. 0~100까지 숫자중 홀수만 출력하시오(함수).

```
int printOddNumber(const 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;
}
```

```
 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
```

3. 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;
}
```

3. 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

3. 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.

3. 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.

3. 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..

3. 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.

3. 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;
4   }

    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;
}
```

3. 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.

3. 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;
}
```

3. 문제 풀이

7. 대소문자를 전환하는 프로그램에 함수 개념을 적용하여 풀어보세요.

```c
#include <stdio.h>

int transeCharactor(char c);

int main(void)
{
    char sec_char;

    printf("글자를 한 개 입력하세요 : ");
    scanf("%c", &sec_char);

    transeCharactor(sec_char);

    return 0;
}

int transeCharactor(char c)
{
    if ((c < 91 && c > 64) || (c > 96 && c < 123))
    {
        printf("입력한 글자는 = %c\n", c);
        printf("아스키 코드로 표현하면 = %d\n", c);


        printf("원래 : %c, 변환 후 : %c \n", c, c ^ 0x20);

        return 1;
    }


    return 0;
}
```

풀이 : char 타입의 변수 한개를 입력받아 범위 확인을 한뒤 0x20과 XOR 연산을 통해 대소문자 변환을 진행한다.

```
chh@chh-15ZD970-GX7SK:~/proj/es02/Lv01-02/HyunhoCha/homework/c/03$ ./a.out
글자를 한 개 입력하세요 : z
입력한 글자는 = z
아스키 코드로 표현하면 = 122
원래 : z, 변환 후 : Z
chh@chh-15ZD970-GX7SK:~/proj/es02/Lv01-02/HyunhoCha/homework/c/03$ ./a.out
글자를 한 개 입력하세요 : Z
입력한 글자는 = Z
아스키 코드로 표현하면 = 90
원래 : Z, 변환 후 : z
chh@chh-15ZD970-GX7SK:~/proj/es02/Lv01-02/HyunhoCha/homework/c/03$
```

3. 문제 풀이

8. 달력을 계산하는 프로그램을 만들어보세요

```
void printCalendar(int year, int month)
{
    int i;
    unsigned int dayCount = 0;
    char daySpace = 0;
    char day;
    char space = 0;

    if (isLeapYear(year))
    {
        g_month_day[1] = 29;
    }


    for (i = 1; i < year; i++)
    {
        if (isLeapYear(i))
        {
            dayCount += 366;
        }
        else
        {
            dayCount += 365;
        }
    }

    dayCount += 365;
```

달력을 계산할때는 2가지만 알면 된다
1. 그달의 1일이 몇요일인지
2. 그달의 끝이 몇일인

2번의 경의 길이가 12인 배열을 만들어 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 값들을 저장해두었다 이때 윤년을 확인하여 윤년인 경우에는 코드에서처름 2월달의 일수를 28일에서 29일로 변경하였다.

그다음은 그달의 1일이 몇요일인지 알아내는 것인데 이것은 0년 1월을 기준으로 몇일이 지났는지 날짜를 세어 7로 나눈 나머지만큼 요일을 뒤로 밀면 된다.

3. 문제 풀이

8. 달력을 계산하는 프로그램을 만들어보세요

```
    for (i = 0; i < month - 1; i++)
    {
        dayCount += g_month_day[i];
    }

    daySpace = dayCount % 7;



printf("일\t월\t화\t수\t목\t금\t토\n");
    printf("----------------------------

    for (i = 0; i < daySpace; i++)
    {
        printf("\t");
        space++;
    }


    day = g_month_day[month - 1];

    for (i = 1; i <= day; i++)
    {
        printf("%2d\t", i);
        if (space == 6)
        {
            printf("\n");
            space = 0;
        }
        else
        {
            space++;
        }
    }
    printf("\n");
}
```

0년 1월을 기준으로 현재까지의 요일수를 모두 구한다음 7로 나누어 나머지만큼 칸을 밀어넣는다.

밀어넣은다음부터 1일을 출력하면된다.

```
Insert Year, month : 2021 04
일      월      화      수      목      금      토
-------------------------------------------------------
                                1       2       3
 4       5       6       7       8       9      10
11      12      13      14      15      16      17
18      19      20      21      22      23      24
25      26      27      28      29      30
```

3. 문제 풀이

8. 달력을 계산하는 프로그램을 만들어보세요

```
int isLeapYear(int year)
{
    if (year % 4 == 0)
    {
        if (year % 100 == 0)
        {
            if (year % 400 == 0)
            {
                return 1;
            }
            return 0;
        }
        return 1;
    }

    return 0;
}
```

윤년을 판별하는 코드이다 윤년규칙은 다음과같다.
1. 4로 나누어 떨어지면 윤년
2. 단 100으로 나누어 떨어지면 윤년이 아니다.
3. 그러나 400으로 나누어 떨어지면 윤년.

3. 문제 풀이

8. 달력을 계산하는 프로그램을 만들어보세요

```
char g_month_day[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

int main(void)
{
    unsigned int year = 0;
    unsigned int month = 0;

    printf("Insert Year, month : ");
    scanf("%d %d", &year, &month);

    //exception
    while(1)
    {
        if (year >= 0 && month >= 1 && month <= 12)
        {
            break;
        }

        printf("Insert Year, month : ");
        scanf("%d %d", &year, &month);
    }

    printCalendar(year, month);

    return 0;
}
```

입력을 받는 main문 코드이다. 입력시에 년, 월 범위가 유효한지 검사하고있다.

3. 문제 풀이

9. 0 ~ 3까지 x^2에 대한 정적분을 수행하는 프로그램을 작성하세요.

```c
double integral_method1(const int a, const int b, double (*f)(double x), int resolution)
{
    double result, deltaX;
    int i;
    deltaX = (b - a) / (double) resolution;

    for(i = 0; i < resolution; i++)
    {
        result += f((a + deltaX / 2) + (i * deltaX)) * deltaX;
    }

    return result;
}
```

```c
double integral_method2(const int a, const int b, double (*f)(double x), int resolution)
{
    double result, deltaX;
    int i;
    deltaX = (b - a) / (double) resolution;

    for(i = 0; i < resolution; i++)
    {
        result += (f(a + deltaX * i) + f(a + deltaX * (i+1))) /2 * deltaX;
    }

    return result;
}
```

정적분이라는 것은 쉽게말해 함수 f(x)가 있가 있을때 임의의 구간 a, b를 적분하는 것이다.

임의의 구간이없이 적분하면 부정적분이라 부르며 적분시 적분상수C가 붙는다.

정적분시 구간 a, b구간의 넓이를 구해야하는데 이때 구분구적법을 사용하였다.

구분구적법 사용시 쪼개는 도형의 형태에따라 사각형과 사다리꼴이 있는데 사다리꼴형태의 구분구적법의 정확도가 더 높다.

3. 문제 풀이

구분구적법을 이용하여 적분을 한 결과이다.

```c
int main(void)
{
    int a = 0;
    int b = 3;
    double result = 0.0;

    result = integral_method1(0, 3, function1, 4096);

    printf("integral x^2 : %lf \n", result);

    result = integral_method2(0, 3, function1, 4096);
    printf("integral x^2 : %lf \n", result);
    return 0;
}
```

```
integral x^2 : 9.000000
integral x^2 : 9.000000
```

4. 질문

Q1. c언어는 함수 오버로딩을 지원하지 않나요?