2018.02.23 - 3일차 수업

---

* 매달 정기 시험 (40~100문제)

1. C언어 / 자료구조

(8시간 금요일)

2. 리눅스 시스템 프로그래밍 / 리눅스 커널

(8시간 금요일)

3. 공업수학 / 펌웨어 프로그래밍(MCU) / FPGA

(72시간 금, 토, 일 3일간)

4. OpenGL(그래픽 라이브러리) / 신호처리 / 물리

(72시간 금, 토, 일 3일간)

-> 시험 후 항상 오답노트 작성

---

* github로 과제 제출하는 법

SHL-Education/Homework에서

Fork누르고 아래 링크 누른다

-> e146/Homework 클릭

-> 본인 이름폴더 클릭

파일 올리고 (x일차 최대성)

커밋 끝낸다음

Pull request 꼭 눌러서 진행 해야함

---

* 데이터 타입

int, short, char, float, double, long double

int: 4 byte(32 비트 = 2^32 개를 나타낼 수 있음)

대략 42 억 9 천만

short: 2 byte(16 비트 = 2^16(65536)개에 해당)

char: 1 byte(8 비트 = 2^8(256)개에 해당)

float: 4 byte

double: 8 byte(2^64)

long double: 12 or 16 byte

데이터 타입에 unsigned 가 붙으면 음수가 존재하지 않고 반대로 unsigned 가 없으면 음수값이 존재함

---

* 2진법에서 음수 만드는 방법

기본적으로 숫자 1과 -1을 더하면 0이 나와야 한다.

여기서 주의할 것은 맨 앞의 부호비트만 반전시키면 1에서 -1이 된다고 생각해서는 안된다

8비트로 확인해보자

8bit에서 실제 1값 0000 0001 -> +1

부호비트만 반전한 값 1000 0001

위의 두 값을 더한 값 1000 0010

-> 결과를 보면 0이 아닌 다른 값이 나온다.

올바른 음수를 획득하는 간단한 방법은 다음과 같다

[음수로 바꾸려 하는 2진법 숫자]의 가장 오른쪽에 있는 자릿수를 포함하여 그 위로 모두 반전시키고 나머지는 그대로 둔다. 그러면 그 값이 음수값이다.

여러 2진법 숫자들을 이용하여 확인해보자

0000 0001 +1

1111 1111 -1

---

1 0000 0000 0 (맨 앞의 1 은 버린다)

확인을 해보기 위해서 5 와 10, 28 등에 실험해보자!

0000 0101 +5

1111 1011 -5

---

1 0000 0000 0 (이하 동문)

0000 1010 +10

1111 0110 -10

---

1 0000 0000 0 (이하 동문)

0001 1100 +28

1110 0100 -28

---

1 0000 0000 0 (이하 동문)

---

* 오버플로우

어떤 데이터 타입이 표현할 수 있는 최대값이 있고 이 범위를 벗어날 경우 오히려 맨 아래로 내려가는 현상

* 언더플로우

표현할 수 있는 최소값에서 더 아래로 내려갈 경우

맨 위로 올라가는 현상

ex) char 타입은 -128 ~ 127이므로

127 + 1 = -128

-128 - 1 = 127

127 + 2 = -127

-128 - 2 = 126

---

* 아스키 코드

문자가 숫자로 치환되기 때문에 암호화에 매우 효율적이다.

기본적으로 우리가 일상에서 사용하는 모든 정보는 유선이 아닌 무선을 통해서 보급되고 있다.

(전화, Wi-Fi, LTE 등)

문제는 SDR이라는 무선통신 분야를 이용하면 무선통신을 하는 모든 데이터를 가로챌 수 있다.

때문에 아스키 코드를 이용한 암호화가 필요하다

---

*암호화의 역사

가장 최초로 암호화를 사용한 사람은 로마의 시저 장군이다

군사 기밀 문서를 가로챌까봐 각 모든 문자에 +3을 했고 받은 문서를 다시 -3해서 복호화 하면 암호 해독이 가능했다. (그래서 시저 암호화라는 별명이 생겼다)

세계 1차, 2차 세계대전에서도 이 기법은 계속해서 발전해왔고 지금도 계속 진행중이다.

---

* printf 안에서 '%' 출력

메타문자 방식을 사용하는 방법과

아스키 코드를 이용하는 방법이 있다.

(디버깅 용도로 많이 쓰기 때문에 아무거나 상관없음)

메타문자 방식 '%%' -> '%'로 출력

아스키코드 '%'는 숫자로 37에 해당한다.

그러므로 %를 출력하고자 한다면

%c에 37을 넣어주면 '%'를 출력할 수 있다.

---

* Bit Operator

1. Bit 연산자는 bit 단위로 연산을 수행
2. <<는 왼쪽으로 bit 이동
3. >>는 오른쪽으로 bit 이동
4. >>, <<는 shift 연산자라고 함
5. &는 bit 연산자 and
6. |는 bit 연산자 or
7. ^는 bit 연산자 xor
8. ~는 bit 연산자 not

---

* 논리 연산자 사용시 주의사항!

```
#include <stdio.h>

int main(){

int shortcut1 = 3,   shortcut2 = 0;

int result;


result = (shortcut2 && ++shortcut1);

printf("shortcut1 = %d₩n", shortcut1);


result = (shortcut1 || ++shortcut2);

printf("shortcut2 = %d₩n", shortcut2);시

return 0;

}
```

-> 위의 결과 값

shortcut1 = 3

shortcut2 = 0

-> 이유

&& 연산자 앞의 내용이 거짓이면 && 연산자 뒤의 내용은 무시하고 넘어간다

|| 연산자 앞의 내용이 참이면 || 연산자 뒤의 내용은 무시하고 넘어간다

---

* 숏컷의 이점

기본적으로 if문을 사용하면 mov, cmp, jmp 형식의 3개의 어셈블리 코드가 만들어진다.

shortcut을 사용하면 비교하는 cmp가 사라진다.

특히 ARM으로 구현할 때 더더욱 이득을 볼 수 있다.

(코드 최적화 용도로 사용하는 기법이다)

3일차 수업 과제

http://cafe.naver.com/hestit/55

1, 3, 4, 5, 7, 10, 12번 풀기

---

1. 스키장에서 스키 장비를 임대하는데 돈이 37500원이 든다. 또 3일 이상 이용할 경우 20%를 할인 해준다. 일주일간 이용할 경우 임대 요금은 얼마일까?

(연산 과정은 모두 함수를 이용한다)

```
#include <stdio.h>
int calculateFee(int days);
int main()
{
	int days = 7; //일주일 대여
	int result = calculateFee(days);  //임대 요금
	printf("%d일간 대여시 요금: %d원₩n", days, result);
	return 0;
}

int calculateFee(int days)
{
	if (days < 0)
		return -1;		//계산 실패시
	else if (days < 3)
		return 37500 * days; //3일 미만 대여
	else
		return 37500 * 0.8 *days; //3일 이상
}
```

---

3. 1 ~ 1000 사이에서 3의 배수의 합을 구하시오

```
#include <stdio.h>
int main()
{
	int result = 0;
	for (int i = 1; i <= 1000; i++)
	{
		if (i % 3 == 0)
			result += i;
	}
	printf("1~1000 사이 3의 배수의 합: %d₩n", result);
	return 0;
}
```

---

4. 1 ~ 1000 사이에서 4나 6으로 나눠도 나머지가 1인 수의 합을 출력하라.

```
#include <stdio.h>
int main()
{
	int result = 0;
	for (int i = 1; i <= 1000; i++)
	{
		if (i % 4 == 1 && i % 6 == 1)
			result += i;
	}
	printf("1~1000 사이 4나 6으로 나눠도 나머지가 1인 수의 합: %d₩n", result);
	return 0;
}
```

---

5. 7의 배수로 이루어진 값들이 나열되어 있다고 가정한다. 함수의 인자(input)로 항의 갯수를 받아서 마지막 항의 값을 구하는 프로그램을 작성하라.

```
#include <stdio.h>
int main()
{
	int num;
	printf("7의 배수로 이루어진 배열의 항의 개수 = ");
	scanf_s("%d", &num);
	printf("7의 배수로 이루어진 배열의 마지막 항은 %d입니다.₩n",num*7);
	return 0;
}
```

---

7. C로 함수를 만들 때, Stack이란 구조가 생성된다. 이 구조가 어떻게 동작하는지 Assembly Language를 해석하며 기술해보시오.

esp, ebp, eip등의 Resister에 어떤 값이 어떻게 들어가고 메모리에 어떤 값들이 들어가는지 등을 자세히 기술하시오.

```
int mult2(int num)
{
	return num * 2;
}
int main(void)
{
	int i, sum = 0, result;
	for (i = 0; i < 5; i++)
		sum += i;
	result = mult2(sum);
	return 0;
}
```

---

10. 구구단을 만들어보시오

```
#include <stdio.h>
int main()
{
	printf("구구단₩n");
	for (int i = 1; i < 10; i++)
	{
		for (int j = 1; j < 10; j++)
		{
			printf("%d x %d = %d	"
				, i, j, i*j);
		}
		printf("₩n");
	}
	return 0;
}
```

---

12. 리눅스에서 디버깅 하는 방법을 정리한다.

명령어 창에 gdb debug 라고 입력하면 디버거가 켜지면서 (gdb)창이 보인다. 여기에 b main 을 입력하고 r을 입력하면 main함수에 멈춘다 disas를 입력하여 현재 어셈블리어 코드와 위치가 보이는데 b *[주소]를 이용하여 다른 위치에 Breakpoint를 걸어줄 수 있고 si를 입력하여 한 줄씩 실행시키면서 확인이 가능하다.

---

gdb 상에서 아직 소개하지 않은 명령들

bt	c

이 2 개에 대해 조사해보고 활용해보자 ~

bt -> 오류가 발생한 함수를 역으로 찾아간다.

c -> 다음 브레이크 포인트를 만날 때 까지 계속 수행한다.

활용법

b main 또는 원하는 곳에 브레이크 포인트를 잡고
오류가 발생할 때 까지 c를 통해 진행하면, 세그먼트 폴트 등의 오류가 발생하고 디버그가 멈추는데
여기서 bt 를 통해서 전체 스택 프레임을 확인하고 어떤 함수에서 호출시에 문제가 발생하였는지 확인
단, 일반적인 라이브러리에서는 오류발생 확률이 없다고 보고, 그 함수를 호출시에 문제를 의심한다.
다시 프레임을 이동하면서, 로컬변수와 전역변수 등을 확인하면서 디버깅이 가능하다.