고려대학교 ALPS 2021

C언어 스터디

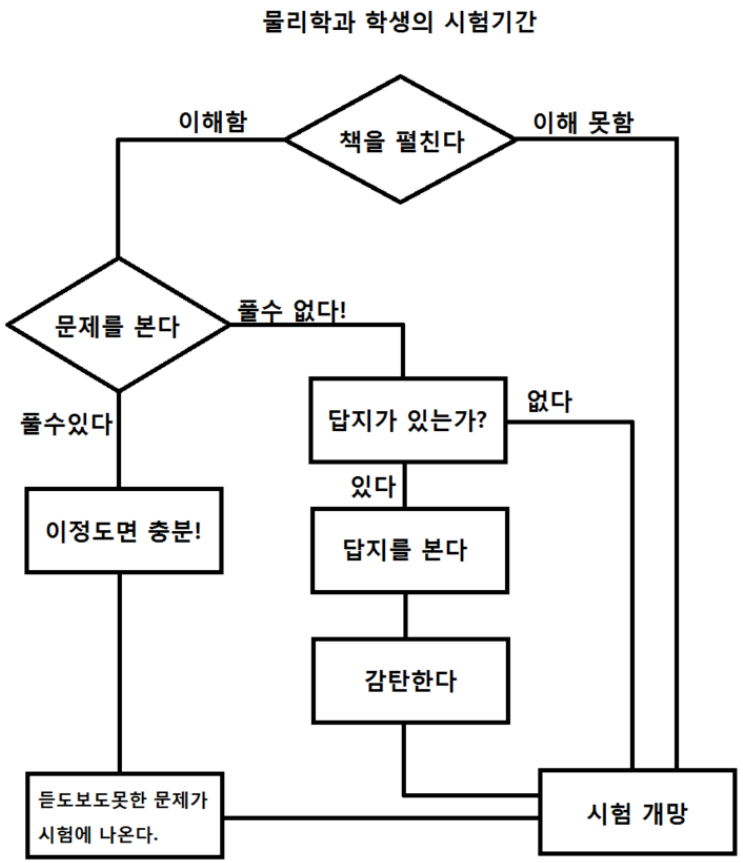
Week 0x02

조건문, 분기문, 반복문,
비트 연산, 아스키 코드

Contents

0x01 Conditional

조건문, 분기문이란?



왼쪽의 그림같이 알고리즘을 묘사한 그림을 보면, (이해함/이해 못함)처럼, **특정 조건**에 따라 프로그램의 **흐름이 나눠지는 것**을 볼 수 있습니다.

이런 역할을 담당하는 문법이 바로 **조건문**, **분기문**입니다. 그 이름처럼 특정 조건에 따라 프로그램의 흐름을 분기로 나눠주는 역할을 합니다.

원래 C의 특성은 코드가 아래로 차례대로 실행되는 것인데, 이런 흐름을 제어한다는 의미에서 조건제어문이라고도 합니다.

0x01 Conditional

조건문, 분기문이란?

if-else

조건문은 크게 if-else, switch-case, **삼항 연산자** '?' 세가지입니다.
if문은 아래와 같이 구성되어 있습니다.

```
if(조건 a){
    // A

}
else if(조건 b){
    // B

}
else if(조건 c){
    // C

}
else{
    // D

}
```

if(조건 1){} 이라는 코드는, '만약' 이라는 if의 의미에 맞게 조건 1이 만족된다면 {}안의 코드가 실행된다는 의미입니다.
if의 짝으로 else if, else 두가지가 있는데요, else if는 그 이전에 있던 if 문의 조건이 충족되지 않으면, else if 안의 조건을 검사하여 코드를 실행한다는 뜻입니다. else if는 왼쪽과 같이 여러 개가 들어갈 수 있습니다.
else는 '그렇지 않으면' 이라는 의미에 맞게 위의 조건들 중 만족하는 것이 하나도 없을 때 실행합니다.

즉, 왼쪽의 코드는, 조건 a를 만족하면 A, 만족하지 않을 시 b를 검사하여 만족한다면 B, 아니면 c를 검사하고, 만족하면 C, 아니면 D를 실행합니다.

이 모든 과정은 순차적으로 실행되므로, 조건을 만족하는 경우가 있다면 해당 중괄호 안의 구문의 코드를 실행하고 전체 블럭을 빠져나옵니다.

예를 들어, a, b를 동시에 만족하는 코드는 A만 실행되고, B는 실행되지 않습니다.

0x01 Conditional

조건문, 분기문이란?

if-else

if – else 안에 새로운 if – else를 넣는 것도 가능합니다.

```
if(조건 a){
  printf(”a”);

}
else {
  if(조건 b){
    printf(“b”);
  }
  else{
    if(조건 c){
      printf(“c”);
    }
  }
}
```

이것을 조건문의 중첩이라고 합니다.

코드를 이해하기가 조금 어려울 수도 있는데, 이것은 나중에 배울 논리 연산자로 간단하게 만들 수 있습니다.

그런데, 코드의 흐름을 생각해보니 else if의 역할과 비슷하다는 생각이 들지 않나요?

0x01 Conditional

조건문, 분기문이란?

if-else

만약 조건이 만족될 때 실행되는 중괄호 안의 코드 구문이 한 줄이라면 중괄호를 생략하는 것도 가능합니다. 즉, 중괄호가 없다면 구문 바로 다음 줄만 구문의 일부로 인식합니다.
예를 들어, 아래의 두 코드는 같습니다.

```
if(조건 a){
    printf("a");

}
else {
    printf("b");
}
```

```
if(조건 a)
    printf("a");

else
    printf("b");
```

이러한 관점에서 볼 때,
else if 구문은
if-else의 중첩으로 볼 수 있습니다.
즉, 왼쪽을 간단하게 하면 오른쪽입니다.

```
if(조건 a){
  printf("a");

}
else {
  if(조건 b){
    printf("b");
  }
}
```

```
if(조건 a)
    printf("a");

else if(조건 b)
    printf("b");
```

More
그러나 무조건적인 중괄호 생략은 코드 이해에 어려움을 주므로 추천드리지 않습니다. 저번 시간에도 말했듯, 코드의 가독성 또한 중요합니다.

0x01 Conditional

조건문, 분기문이란?

if-else

그렇다면,
퀴즈를 한 번 풀어봅시다.
조건 a, b를 만족하며,
c는 만족하지 않을 때

오른 쪽 두 코드를 실행하면
결과가 어떻게 나올까요?


```
if(조건 a){
    printf("A");

}
else if(조건 b){
    printf("B");

}
else if(조건 c){
    printf("C");

}
else{
    printf("D");

}
```

```
if(조건 a){
    printf("A");

}
if(조건 b){
    printf("B");

}
if(조건 c){
    printf("C");

}
else{
    printf("D");

}
```

0x01 Conditional

조건문, 분기문이란?

정답

- 첫번째 코드
 첫번째 코드는 조건 a를 만족하여 A가 출력되며, a를 만족했기 때문에 else문, 즉 else if 문은 실행되지 않기 때문에, A를 출력하고 코드가 끝납니다.

 >> A

- 두번째 코드
 두번째 코드는 조건 a를 만족하여 A가 출력된 후, 다음 if문으로 가서 조건 b를 만족하므로 B가 출력되고, 그 다음 if문에서 조건 c를 만족하지 않으므로 else문이 실행되어, D를 출력하고 코드가 끝납니다.

 >> ABD

0x01 Conditional

조건문, 분기문이란?

Switch-case

Switch문은 분기마다 각각의 case를 나누어 처리합니다.
조건식을 만드는 방법도 if문과 다르고, 오직 조건이 되는 변수가 특정 조건과 같은지 판단하는,
동등 조건만 판별할 수 있습니다.

```
switch(num){
  case a:
    A;
    break;
  case b:
    B;
    break;
  default:
    C;
    break;
}
```

왼쪽이 그 예시인데, 중괄호로 둘러싸여 있던 if문과 다르게 ':'와
'break;'가 보입니다.

switch 문은 위에서부터 아래로 순차적으로 실행됩니다.(C의 특징)
그리고 각 case를 만나면, num이 case와 같은지 검사합니다.
즉, 'case a:'는 num이 a와 같은지 검사하겠다는 말이며, num이 a와 같
다면, 콜론 아래의 들여쓰기 된 코드들을 차례대로 실행합니다.
break;의 의미는 현재 들어와있는 중괄호를 탈출하겠다는 의미입니다.
즉, 아래의 코드는 더이상 실행하지 않고 switch문을 탈출하는 것입니다.

default는 말 그대로 기본, 즉 위의 case에 모두 해당하지 않을 때임
즉, if-else if-else 처럼 작동함

0x01 Conditional

조건문, 분기문이란?

Switch-case

만약 break가 없다면 어떻게 될까요?

break가 없다면, switch문을 빠져나오지 못하므로 그 아래의 case들을 모두 실행하게 됩니다.

아래의 예시에서 a를 만족한다면 b의 만족, 불만족과 상관없이 ABC 모두 실행하게 되는 것이죠.

```
switch(num){
  case a:
    A;
  case b:
    B;
  default:
    C;
}
```

이렇게 break로 중단하지 않고 그 다음 case, default가 계속 실행되는 상황을 '구멍에 떨어지다'에서 유래된 "fall through"라고 부릅니다.

그렇다면, default는 어차피 마지막 순서에 위치하므로, default 아래에는 굳이 break;를 적어주지 않아도 되겠죠?

0x01 Conditional

조건문, 분기문이란?

Switch-case

break가 없는 상황 응용 예시

정수 num이 2 또는 3인지 판별

```
switch(num){
  case 1:
    printf(“no”);
    break;
  case 2:
  case 3:
    printf(“yes”);
    break;
  case 4:
    printf(“no”);
    break;
  default:
    printf(“넷 다 아님”);
}
```

0x01 Conditional

조건문, 분기문이란?

삼항 연산자

삼항 연산자 Ternary operator는 조건에 따라 간단히 값을 반환하는 연산자입니다.

이름에 맞게 세가지 항을 필요로 하며, 예시는 아래와 같습니다.

`조건 ? A : B;` 물음표 이전의 조건이 맞다면 A, 아니면 B를 반환하게 됩니다.

if-else의 간편화라고도 볼 수 있겠죠.

예를 들어, a와 b중 큰 것을 c에 대입하고 싶다면, 아래와 같이 하면 됩니다.

```
int c = a > b ? a : b;
```

A가 b보다 크다면 조건을 만족하므로 a가 c에 대입되고,
작거나 같다면 조건을 만족하지 않으므로 b가 대입됩니다.

그렇다면 이런 조건식을 어떻게 표현할지, 지난 시간에 언급했던 논리 연산자, 관계 연산자를 통해 알아봅시다.

0x01 Conditional

논리 연산자, 관계연산자

먼저 지금까지 말한 '조건'의 값은 보통 true, false로 나타냅니다.
국어 모의고사 지문 같은 곳에서 보던 명제의 참, 거짓을 나타내는 그것과 같습니다.
이것을 보통 Java 등의 언어에서는 **불리언 타입** Boolean type, 줄여서 Bool 타입이라고 합니다.

C언어에서는 이러한 bool type이 없기 때문에 0을 false, 0이 아닌 값을 true로 나타냅니다.

관계 연산자는 이전 슬라이드에서 본 것과 같이, 이미 수학 시간에 많이 봤던 부등호들을 말합니다. 다른 점이 있다면, C에서 등호 '=' 는 대입 연산자이므로 등호를 두 번 사용하여 ==로 표현합니다. 다음은 관계 연산자의 목록입니다.

`<, >, <=, >=, ==, !=` 부등호와 등호를 같이 쓴 연산자 <=, >=는 ≤,≥를 의미하며,

!=는 ≠를 의미합니다.

느낌표 !는 사실 부정의 의미와 같습니다. 그렇기에 !=가 같지 않음을 나타내는 것이죠.
예를 들어 (2<3) 은 true를 나타내지만, !(2<3)은 false를 나타냅니다.

0x01 Conditional

논리 연산자, 관계연산자

그렇다면 논리 연산자는 무엇일까요?

참, 거짓인 명제들을 다루는 학문을 보통 논리학이라고 하죠.
여기서 만들어진 연산이 바로 논리 연산자입니다. true와 false를 이용해 연산을 진행합니다.

논리 연산자는 세가지, AND, OR, NOT을 나타내는 연산이 있습니다.
AND, OR, NOT 게이트를 들어보신 분들이 있다면 이해가 쉬울 것입니다.

AND를 나타내는 연산자는 &&로, 양 옆의 식이 모두 true일 때만 true입니다.
즉, 2<3 && 4<5는 true이지만, 2<3 && 5<4는 false입니다.

OR을 나타내는 연산자는 ||로, 양 옆의 식이 하나라도 true이면 true입니다.
즉, 2<3 || 4>5는 true이지만, 2>3 || 5<4는 false입니다.

NOT은 !를 이용하며, 아까 말했듯이 !(true)이면 false, !(false)이면 true입니다.

0x01 Conditional

논리 연산자, 관계연산자

논리 연산자를 다르게 표현하자면 집합의 논리로 표현할 수 있습니다.
꼭 이해하실 것은 아니고 이렇게 이해하면 쉽습니다.

AND는 교집합, OR은 합집합, NOT은 여집합을 나타냅니다.

집합 A = { x ¦ x<5}
집합 B = { x ¦ x>3}
집합 C = { x ¦ x>7} 이라고 한다면,

(x<5) && (x>3) 은, x가 5보다 작고 3보다 클 때만 true인데, 이것은 A와 B의 교집합입니다.

(x<5) ¦¦ (x>7) 은, x가 5보다 작거나 7보다 클 때 true인데, 이것은 A와 C의 합집합입니다.

!(x<5)는 x가 5보다 크거나 같을 때 true인데, 이것은 A의 여집합입니다.

이러한 수학적 연산과 논리 연산을 0과 1을 나타내는 비트에 더하여
2학년 전공 논리설계, 이산수학 등에서 배우실 텐데,
지금은 이렇게 알아두면 이해가 쉽다는 것만 아시면 될 것 같습니다!

0x02 Iteration

반복문이란?

조건문에서는 조건에 따라 프로그램의 분기가 갈렸었죠.

반복문이란, 이름 그대로 조건을 만족하는 동안 특정 코드를 계속해서 반복할 수 있는 문법입니다. 반복문 또한 프로그램의 흐름을 제어하므로 반복 제어문이라고도 합니다.

그렇다면 특정 코드를 여러 번 적는 번거로운 행위를 피할 수 있고, 거기에 더해 특정 조건을 만족할 때 까지 계속해서 코드를 실행할 수도 있겠죠.

조건문에서 가장 편리하고, 유용하며, 자주 사용되는 구문은 if-else 구문입니다.
반복문 문법의 종류에는 for문, while문, do-while 문이 있는데요,
그 중 가장 편리하고, 유용하며, 자주 사용되는 구문은 for문이 있습니다.

다음 슬라이드에서 정확히 알아보도록 합시다.

0x02 Iteration

반복문이란?

```
for(초기식; 조건식; 증감식;){
  // 반복할 코드;
}

for(int i=0; i<30; i++){
  printf("무야호\n");
}

int i = 0;
for(; i<30; i++){
  printf("무야호\n");
}

for(int i=0; i<30;){
  printf("무야호\n");
}
```

for문은 복잡해보일 수도 있으나 순서를 정확하게
이해하신다면 정말 유용한 도구가 될 수 있을 것입니다.☺️
두번째의 무야호 예시와 함께 보면 편하실 겁니다.

for문의 순서는 다음과 같습니다.
1. 초기식이 있다면 초기식을 실행(int i=0;)
2. 조건식이 있다면 조건식을 검사,
   조건식이 참이라면 반복할 코드를 실행하며,
   거짓이라면 반복을 종료합니다.
3. 조건식이 참이라 코드가 실행되고 나면, 증감식을
   실행하고 다시 2번으로 돌아갑니다.

증감식은, 코드가 영원히 반복되지 않기 위해 조건에
변화를 주는 식입니다. 예시를 보시면 저번 시간에 배웠던
증감 연산자가 붙어있습니다.

위의 순서대로 본다면, 두번째 무야호는 i=0에서 29까지 총
30번 무야호를 외치고, 마지막 반복에서 i++를 실행하면,
i=30이 되어 조건 식을 거짓으로 만들게 되어 반복을 종료

0x02 Iteration

반복문이란?

More
각각의 반복을 iteration 또는 loop라고 합니다.
영어로 말하면 있어보여요.

```
for(초기식; 조건식; 증감식;){
  // 반복할 코드;
}

for(int i=0; i<30; i++){
  printf("무야호\n");
}

int i = 0;
for(; i<30; i++){
  printf("무야호\n");
}

for(int i=0, j=1; i<30;){
  printf("무야호\n");
}
```

근데 이전의 순서를 보면, 초기식, 조건식, 증감식 등이 '있는' 경우라고 정의를 했습니다.
즉 이런 식들은 세번째, 네번째 무야호처럼 생략될 수 있습니다.

세번째 예시처럼 초기식을 따로 뺀 후 초기식을 생략하고 세미콜론만 적을 수도 있고, 네번째 예시처럼 초기식에 콤마 ','을 통해 여러 개를 동시에 초기화할 수도 있습니다.

또한, 4번째 예시를 보면 증감식이 생략되어 있습니다.
만약 증감식이 생략되어 있다면, i=0에서 계속 변화하지 않으므로 i<30을 영원히 만족하게 되어,

흔히들 말하는 '무한루프'에 빠지게 되겠죠.
(각각의 반복을 iteration 또는 loop라고 합니다.)
그러면 4번째 코드는 미친듯이 무야호를 외칠겁니다.

그만큼 신나겠죠..?

0x02 Iteration

반복문이란?

More
초기식에 들어간 i, j처럼 값이 변화하며 조건을 결정하는 값을 iterator라고 하는데, 이런 iterator의 이름을 정하는 관행은,
선형대수학에서 배우는 standard vector 이름처럼 i,j,k 순입니다.

More
여기서도 중괄호가 없으면 바로 다음 코드만 구문의 일부로 인정한다는 규칙은 같습니다.

```
for(int i=0; i<30; i++)
  printf("무야호\n");

for(int i=0; i<10; i++){
  printf("무야호\n");
  for(int j=0; j<10; j++){
    printf("그만큼 신나시는거지\n");
  }
}
```

반복문 또한 조건문처럼,

중괄호를 생략할 수도 있고,

중첩하여 여러 반복문을 넣는 것도 가능합니다.

아래의 중첩된 반복문을 실행하면
(무야호+ 그만큼 신나시는거지*10)이 10번 출력되겠죠?

0x02 Iteration

반복문이란?

```
while(조건식){
  // A;
}

do{
  // A;
} while(조건식)
```

while문은 for문보다는 조금 더 직관적이라고 할 수 있습니다. '~동안'이라는 영어 단어의 뜻 그대로 조건식이 true인 동안 계속해서 A를 실행합니다. 따로 증감식이 정의되어 있지 않기 때문에, 직접 개발자가 조건문을 변화시키도록 증감식을 만들어줘야 합니다. 그렇지 않으면 무한루프에 빠지게 됩니다.

do-while 문 또한 이름 그대로, 한 번 실행을 한 후, while을 검사합니다. 즉, while문의 순서가 뒤집힌 것이라고 볼 수 있습니다. A를 실행한 뒤, 조건식을 검사하여 true이면 다시 반복, 아니면 반복을 종료합니다. while처럼 증감식을 따로 정의해줘야하는 점에서는 같으나, 조건식의 참, 거짓과 상관 없이 최소한 한 번 실행된다는 점이 다릅니다.

그런데, '무한 루프'는 대체 어떻게 활용할까요?
switch-case에서 배운, break를 통해 활용해봅시다.

0x02 Iteration

또 다른 흐름 제어 구문

More
return에 대해서 먼저 알고 싶으시다면, 함수를 종료하고, 값을 반환하는 문법이라고 알아둡시다

```
for(int i=0;i<5;i++){
  if(i==3){
    break;
  }
}

while(1){
  int input;
  scanf("%d",&input);
  if(!input) break;
}
```

for, while, if 등의 제어문 말고도 코드의 흐름을 제어하는 문법으로 아까 배웠던 break; continue; return; 등이 있습니다. 이 중 return은 우리가 짠 코드의 가장 아래에도 있는데, 이것은 함수를 공부할 때 배워보도록 하겠습니다.

먼저 **break**입니다.
switch-case에서도 말했듯이, 현재 코드를 감싸고 있는 중괄호를 깨부시고 탈출하는 문법입니다. 그렇기에 switch문의 case를 탈출하는 데 쓰였고, for, while문 등에서도 for, while을 탈출하는 데 쓸 수 있습니다. 특히 무한루프에서도 탈출할 수 있습니다.

첫번째 예시에서 i=3이 되면 for 반복문을 종료하게 됩니다.

두번째 예시에서는 조건문이 1, 즉 항상 true이므로 이 프로그램은 무한 루프를 돌면서 계속해서 입력을 받는데, 조건식이 !input 이므로, input이 false일 때, 즉 input이 0일 때 break를 통해 무한 루프를 종료합니다.

0x02 Iteration

또 다른 흐름 제어 구문

```
for(int i=0;i<5;i++){
  if(i==3){
    continue;
  }
  printf(“%d”,i);
}
```

다음은 continue입니다.

continue는 계속한다는 의미에 맞게, 반복문 전체가 아니라 이번 반복 단 '한 번'을 종료하고, 다음 단계로 넘어갑니다.

여기서 다음 단계란 for/while에서 각각 다릅니다.

for문에서 반복의 다음 단계는 증감식입니다. 그러므로 for문에서 continue를 만나면 continue 아래의 코드를 실행하지 않고 바로 증감식을 실행합니다. 이후 조건문이 참이면 다시 반복을 계속하고, 아니면 종료합니다.

while에서 반복의 다음 단계는 조건식이므로, continue 아래의 코드를 실행하지 않고 바로 조건을 검사합니다.

왼쪽의 예시의 출력은 1245가 되겠죠??

0x03 Bitwise operation

비트 연산

저번 주에 컴퓨터가 데이터를 저장하는 방식인 이진법과 비트를 배웠습니다.

이렇게 데이터를 표현하는 각각의 비트를 조작하는 방법이 있는데, 이것이 바로 **비트 연산자**라고합니다.

이런 비트 연산은 어디에 사용될까요?

비트 연산은 1바이트, 4바이트, 8바이트 처럼 묶인 비트에 접근하여 연산하는 일반 연산자보다 비트 자체에 접근하기 때문에 매우 빠르게 연산이 가능하며, 짧은 코드를 작성할 수 있고, 효율적 메모리 사용이 가능하다는 장점이 있으며, 큰 차이가 없어 보일 수도 있으나 연산의 개수가 많아질수록 효율적입니다.

이러한 장점을 활용하기 위해 비트 하나 하나에 데이터를 저장하는 비트 마스크라는 알고리즘 테크닉에 사용 가능합니다.

0x03 Bitwise operation

비트 연산자

그렇다면 비트 연산자에는 무엇이 있을까요?

비트 연산자는 `&, |, ~, ^, <<, >>` 총 5가지가 있으며,

각각 AND, OR, NOT, XOR 연산자, 그리고 시프트 연산자입니다.

각각 논리 연산자에서 봤던 의미대로,

AND는 둘 다 1일 때 1,

OR은 둘 중 적어도 하나가 1일때 1,

NOT은 비트 반전,

XOR은 비트가 다를 때만 1을 반환합니다.

0x03 Bitwise operation

비트 연산자

각 경우의 수는 다음과 같습니다.

연산자	비트1	비트2	결과
&	0	0	0
	0	1	0
	1	0	0
	1	1	1
|	0	0	0
	0	1	1
	1	0	1
	1	1	1
^	0	0	0
	0	1	1
	1	0	1
	1	1	0

AND/ OR/ NOT/ XOR 연산자는
값을 비트 단위로 나열한 뒤 각각 다음과 같은 연산을 수행합니다.

& (비트 AND)
두 비트가 모두 1이면 1을 반환합니다.
즉, 둘 다 1인 경우에만 1을 반환합니다.

| (비트 OR)
두 비트 중 하나라도 1이면 1을 반환합니다.
즉, 둘 다 0인 경우에만 0을 반환합니다.

~ (비트 NOT)
단순히 비트를 바꿉니다.
즉, 0이면 1, 1이면 0을 반환합니다.

^ (비트 XOR)
두 비트가 다르면 1을 반환합니다.
즉, 두 비트가 같으면 0을 반환합니다.

0x03 Bitwise operation

비트 연산자

시프트 연산자 *Shift operator* 는
오른쪽 피연산자의 수만큼 비트를 왼쪽(<<) 또는 오른쪽(>>)으로 이동시킵니다.

왼쪽으로 이동시키는 경우 비워진 오른쪽 비트는 0으로 설정됩니다.
오른쪽으로 이동시키는 경우 비워진 왼쪽 비트는 부호 비트를 복사하여 채워집니다.

또한, 비트 크기 한도를 넘어 시프트 연산을 하면, 제한을 넘는 오버플로우 *overflow* 또는 언더 플로우 *underflow* 현상으로 인해 정보가 손실됩니다.
예를 들어, 4비트짜리 자료형이 있을 때, <<4 (4번 왼쪽 시프트)연산을 하면 항상 0입니다.

시프트 연산의 예시는 다음과 같습니다 .

```
3(0011) << 2 == 12(1100)
6(0110) >> 2 == 1 (0001) // 비트 정보가 소실될 수 있다.
```

0x03 Bitwise operation

비트 연산자

그런데 생각을 해봅시다.

십진수에서 오른쪽에 0을 붙인다는 것은 10의 거듭제곱을 곱한다는 것과 같죠.

그리고 100 등의 수에서 오른쪽으로 이동시킨다는 것은, 0을 뗀다, 즉 10의 거듭제곱으로 나눈다는 것과 같습니다.

이 개념을 이진수에 적용해보면,

<< 연산은 수에 2의 거듭제곱을 곱하는 연산과 같습니다.
즉, a << i는 $a \times 2^i$와 같습니다.

>> 연산은 수에 2의 거듭제곱을 나누는 것과 같습니다. (비트 정보 손실이 없을 때)

이런 것들이 바로 비트연산에 사용되는 테크닉들입니다.

0x03 Bitwise operation

비트 연산자

간단하게 예시만 들어보도록 하겠습니다.

비트 마스크 *bit mask*는 비트를 사용해 마스크처럼 특정한 부분은 가리고 특정한 부분은 볼 수 있도록 만든다는 느낌입니다.

한 데이터에서 특정 데이터가 있는 비트에 1과 &연산을 하면 특정 비트가 1이면 1, 아니면 0
예를 들어, data & (1<<i) 는 data라는 변수의 오른쪽에서 i번째 비트가 1인지 확인합니다.

| 연산을 한다면 특정 비트에 비트를 쓰는 것도 가능합니다.
예를 들어 data |= 1<<i 라고 한다면, data 변수의 오른쪽 i번째 비트가 0이라면 1로 바꿉니다. (대입 연산자 사용)

이런 연산은 시험문제로 나오거나, 알고리즘 공부를 심도있게 하시면 볼 일이 많으실 겁니다.
제 경험 상 알고리즘 과목 과제, 중간고사에서도 비트 연산을 다룬 바 있습니다.

또 다른 간단한 예시로 홀수/ 짝수 판별을 짧고 빠르게 하자면,
if(num & 1) 이라고 하면, if(num%2 ==1)보다 짝수 판별을 간단하고 빠르게 할 수 있겠죠.

0x04 ASCII Code

아스키 코드

저번 시간에 정수, 실수만 다루고,
문자와 문자열은 이번 주에 다뤄보기로 했었죠.

컴퓨터에서는 데이터를 0과 1의 조합으로만 저장하는데, 어떻게 문자를 표현할 수 있을까요?

0과 1의 조합으로는 수를 표현하기가 제일 편하기에,
C에서는 0에서 127까지의 총 128가지의 수가 각 문자를 표현하도록 합니다.

128? 어디서 봤던 것 같은데,..
맞습니다. Char 자료형의 범위가 -128~127이었습니다.
그래서 char을 이용해 아스키 코드 전체, 즉 문자를 표현할 수 있습니다.

아스키 코드에는 간단하게 영어 알파벳 대소문자와 일부 기호, 이스케이프 시퀀스들이
포함되어 있습니다.

또한 특정한 순서를 가지고 있고, 숫자로 표현되므로 연산까지 가능합니다.
예를 들어, 'A'는 65의 아스키 코드를 갖는데, 'A'+1 == 'B' == 66 입니다.
또한, 이러한 아스키코드 문자는 작은 따옴표로 감싸줘야 합니다.

More
Char은 문자를 나타내는
단어 character의
약자입니다.

0x04 ASCII Code

아스키 코드, 헷갈리는 점들.

아스키 코드 역시 외울 필요는 없고,
알파벳 a가 97, A가 65인 것을 '알아 두면' 좋긴 한데, 몰라도 쓸 수 있습니다.

예를 들어, char 형의 data가 알파벳 소문자일 때, 대문자로 만드는 코드를 짠다면
if(data >= 'a' && data <= 'z') data += 'A' – 'a'; 라고 짤 수 있겠죠.

또한 그냥 수를 문자에 대치시킨 것이기에, char 형이 아닌 다른 정수형으로 문자를 나타낼 수도 있습니다.

예를 들어,
char data1 = 'a';
int data2 = 97;
printf("%c %c", data1, data2); 는 a a를 출력합니다.

문자들이 모인 집합체인 문자열은,
다음 주에 배열을 통해 다루도록 하겠습니다!

0x05 Practice

실습

예시 -
https://www.acmicpc.net/problem/10871
https://www.acmicpc.net/problem/4153

CLASS 기능!
https://solved.ac/class

0x06 Assignment

과제

https://www.acmicpc.net/problem/11654 - 11654: 아스키코드, 아스키 코드
https://www.acmicpc.net/problem/2753 - 2753: 윤년, 조건문
https://www.acmicpc.net/problem/2439 – 2439: 별 찍기 - 2, 반복문
https://www.acmicpc.net/problem/1110 - 1110: 더하기 사이클, 반복문 활용
https://www.acmicpc.net/problem/1259 - 1259: 팰린드롬수: 반복문 + 조건문
3문제 이상
이왕이면 저한테 마음껏 질문하시면서 다 풀어보세요!

부족한 첫 수업
들어 주시느라 수고 많으셨습니다!

들어주셔서 감사하고, 질문과 밥약은 언제든 환영이에요

다음 주에 봅시다~