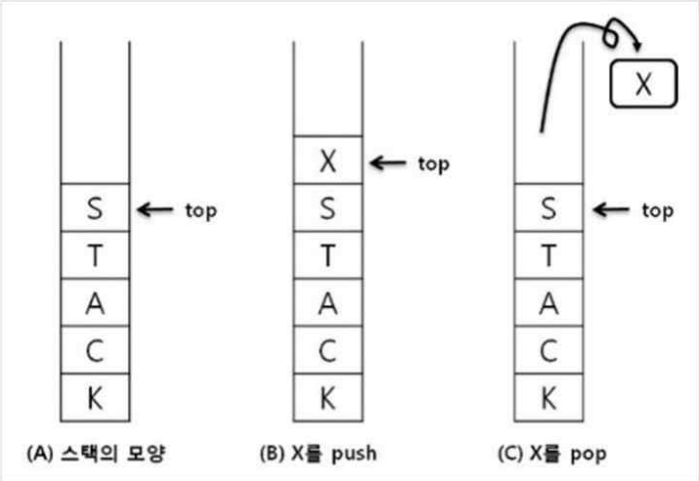
과목명	자료구조 및 알고리즘	분반	x	담당교수	김화성 교수님
학과	전자통신공학과	학번	2016707079	이름	하상천
H/W 3: Decimal to Binary Conversion					

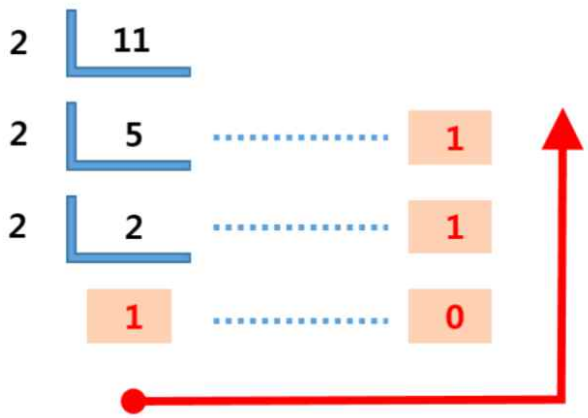
1, 과제설명 (과제에 대한 설명 및 목표)

Design the algorithm which converts a decimal number into a binary number using stack. And Implement your algorithm using C Programming language.

2, 이론 (과제와 관련한 일반적인 내용)



3. 알고리즘 및 자료구조 설계 내용 기술



2로 나눈 나머지를 순서대로 stack에 입력하고 마지막에 몫이 1이 되었을 때 그 값도 stack에 입력해준다. stack에서 하나씩 값을 꺼내어 출력해준다.

4. 소스코드 설명 (직접 작성한 소스코드중에 핵심 부분을 발췌하여 설명)

```
Stack *pStack = (Stack*)malloc(sizeof(Stack));
```

Stack 크기만큼 동적 메모리 할당을 받는다. malloc이 반환하는 주소의 타입은 void * 이므로 Stack * 으로 형변환 시킨다.

```
pStack->stack = (element*)malloc(size * sizeof(element));
```

element 크기에 입력받은 size 값을 곱하고, 그 크기만큼 동적 메모리 할당을 받는다. malloc이 반환하는 주소의 타입은 void * 이므로 element *으로 형변환 시킨다.

```
while (true) {
if (num == 1) {
push(pStack, num);  //마지막 숫자1 스택에 입력
break;  //while 반복문 탈출
}
push(pStack, num % 2);  //2로 나눈 나머지 스택에 입력
num = (num / 2);
}
```

2로 나눈 나머지 값을 stack에 입력하고, 2로 나눈 몫을 다시 num에 대입하여 while문을 반복하였다. 2로 나눈 몫이 1이 되는 경우에 1을 stack에 넣고 break를 통해서 반복문을 탈출하였다.

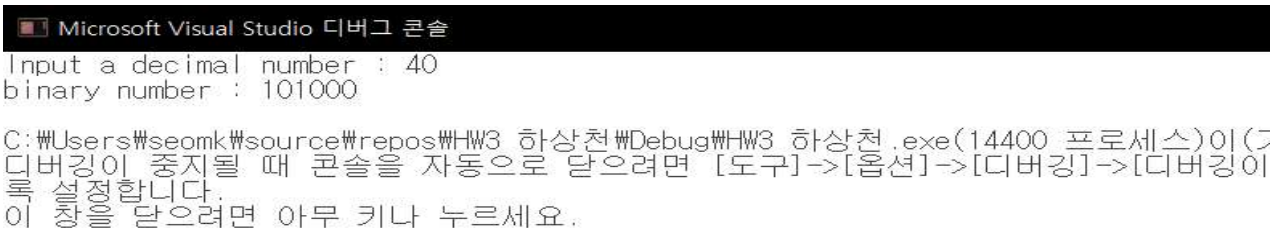
```
while (pStack->top >= 0) {
num = pop(pStack);  //스택에서 값 받아오기
printf("%d", num);
}
```

pop 함수를 통해서 stack에 있는 값을 가져온 후 출력한다. top 이 -1이 되면 stack이 빈 상태이므로 top이 0보다 작아지면 반복문을 탈출한다.

```
void DestroyStack(Stack *pStack) {
free(pStack->stack);
free(pStack);
}
```

stack 먼저 동적 메모리를 반납하고 그 다음 pStack도 동적 메모리를 반납한다.

5. 실행결과 및 설명 (실행 결과를 캡쳐하여 첨부한 후 설명)



(그림을 문서에 포함, 글자처럼 취급 옵션, 잉크 절약과 잘 보이게 하기위해 그림 반전)

십진수를 입력 받고, 이진수로 바꾼 결과 화면입니다.

6. 고찰 (과제를 진행하면서 배운점 이나, 시행 착오 내용, 기타 느낀점)

이번 과제는 스택을 이용하여 십진수를 이진수로 바꾸는 알고리즘을 설계하고, C언어를 통해 구현하는 것이었다. 이진수를 십진수로 바꾸는 알고리즘은 쉽게 생각이 났는데, 십진수를 이진수로 바꾸는 알고리즘이 쉽게 생각이 나지 않았다. 그래서 숫자를 하나하나 노트에 적고 고민해보았더니 알고리즘을 알 수 있었다. 십진수 num을 2로 나눈 나머지를 stack에 입력하고, 2로 나눈 몫을 다시 num으로 저장하였다. 또 num을 2로 나눈 나머지를 stack에 입력하고, 2로 나눈 몫을 다시 num으로 저장하였다. 이렇게 반복하다가 2로 나눈 몫이 1이 되면 1도 stack에 입력하고 반복문을 탈출하였다. 스택의 특성상 LIFO(Last-In First-Out)이기 때문에 이번 과제에 적합한 알고리즘 인 것 같다. 만약 이번 과제를 큐를 이용해서 구현했다면, 2로 나눈 나머지 값들을 배열에 대입해놓고 마지막에 대입한 값부터 큐에 입력해야 될 것 같다. 왜냐하면 큐는 FIFO(First-In First-Out)의 특성이 있기 때문이다. 하지만 이번 과제에서 큐를 이용하면 안써도 되는 배열을 사용하기 때문에 데이터 낭비가 있고, 연산과정이 더 추가되기 때문에 비효율적이다. 따라서 이번 과제는 스택을 이용하는 것이 더 효율적인 것 같다. 항상 느끼는 것이지만 강의로 스택을 배우는 것과 직접 알고리즘을 설계하고 코딩 하는 것은 많이 다른 것 같다. 이번 과제를 통해서 스택과 더 가까워진 것 같다.

7. 전체 소스코드 (글자크기 9에 줄간격을 120%로 유지하고 한 줄이 너무 길지 않게 작성)

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

typedef int element;
typedef struct {
element *stack;  //array to store elements
int max_size;  //maximum size
int top;   //stack top
} Stack;

Stack* CreateStack(int size) {      //Stack 동적 메모리 할당, size 만큼 stack 동적 메모리할당
```

```
Stack *pStack = (Stack*)malloc(sizeof(Stack));
if (pStack == NULL)
{return NULL;}

pStack->stack = (element*)malloc(size * sizeof(element));
if (pStack->stack == NULL) {
free(pStack);
return NULL;
}
pStack->max_size = size;
pStack->top = -1;
return pStack;
}

void push(Stack *pStack, element item) {
if (pStack->top == pStack->max_size - 1) {
printf("Stack is full");
return;
}
pStack->stack[++pStack->top] = item;
}
element pop(Stack *pStack) {
if (pStack->top < 0) {
printf("Stack is empty");
return 0;
}
return pStack->stack[pStack->top--];
}
element top(Stack *pStack) {
if (pStack->top < 0) {
printf("Stack is empty");
return 0;
}
return pStack->stack[pStack->top];
}

void DestroyStack(Stack *pStack) {
free(pStack->stack);
free(pStack);
}



int main(void) {
int num;
```

```
Stack *pStack = CreateStack(100);
printf("Input a decimal number : ");
scanf("%d", &num);  //십진수입력받기
while (true) {
if (num == 1) {
push(pStack, num);  //마지막 숫자1 스택에 입력
break;  //while 반복문 탈출
}
push(pStack, num % 2);  //2로 나눈 나머지 스택에 입력
num = (num / 2);
}

printf("binary number : ");
while (pStack->top >= 0) {
num = pop(pStack);  //스택에서 값 받아오기
printf("%d", num);
}
printf("₩n");

DestroyStack(pStack);  //동적 메모리 반납
pStack = NULL;

return 0;
}
```

(글자크기는 10으로 유지하고 줄간격도 160%를 유지할 것)