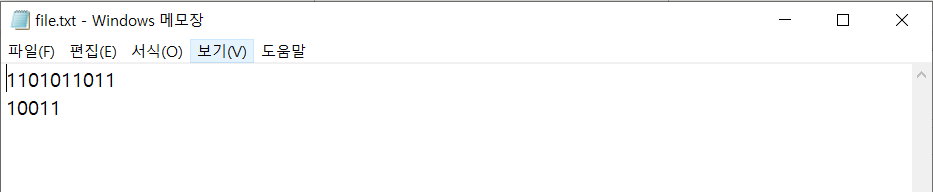
1. File.txt의 두 sequence를 <를 통해 generator에서 표준입력(scanf)로 입력 받고, 두 sequence를 합친 하나의 sequence를 생성한다.



|  |
| --- |
| #include <stdio.h> // generator.c  #include <string.h>  #include <stdlib.h>  int main() {  char n\_bit[20];  char k\_bit[20];  char \*nk\_bit;  int n, k, alt = 0; // length, alter bit  int i;  // input n\_bit, k\_bit from file.txt  scanf("%s", n\_bit);  scanf("%s", k\_bit);  n = strlen(n\_bit);  k = strlen(k\_bit);  nk\_bit = (char\*)malloc(n+k);  // generate n+k bit, output of generator  for (i = 0; i < n; i++)  nk\_bit[i] = n\_bit[i];  for (i = 0; i < k; i++)  nk\_bit[i+n] = k\_bit[i];  printf("%d\n", n);  printf("%d\n", k);  printf("%d\n", alt); // if no alter, alt will be 0  printf("%s\n", nk\_bit);  free(nk\_bit);  return 0; // return n+k bit line  } |

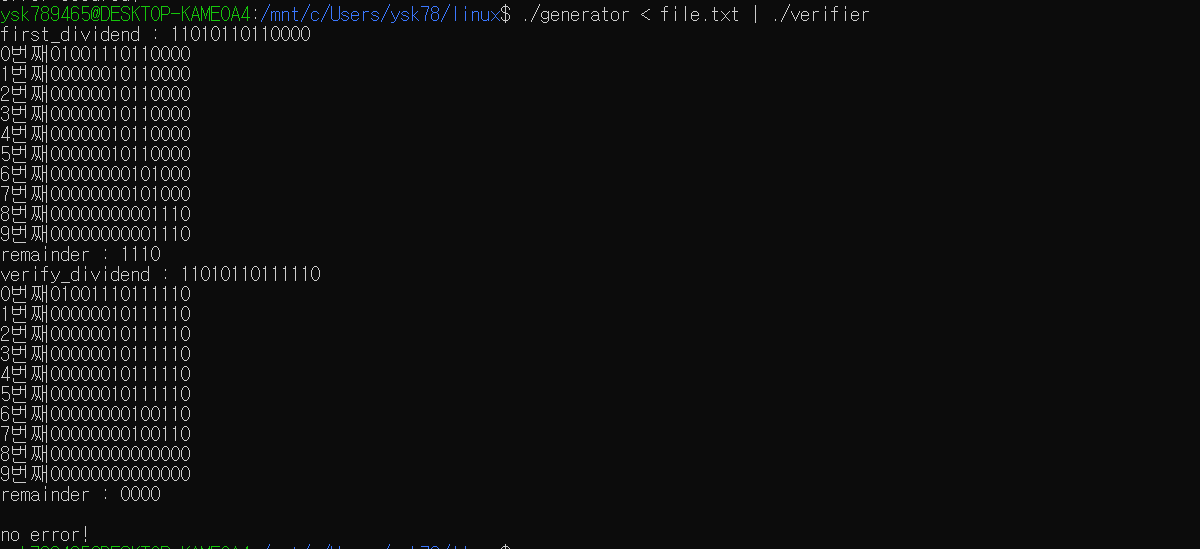
2. alter 프로그램에서는 argument로 입력 받은 위치에 있는 비트를 반전시킨다.

|  |
| --- |
| #include <stdio.h> // alter.c  #include <stdlib.h>  int main(int argc, char\*\* argv) {  int n, k, alt;  char \*nk\_bit;  scanf("%d", &n);  scanf("%d", &k);  scanf("%d", &alt);  nk\_bit = (char\*)malloc(n+k);  scanf("%s", nk\_bit);  // argument error  if (argc != 2) {  printf("argument number error!!\n");  return 1;  }  // alter specific bit  alt = atoi(argv[1]);  if (nk\_bit[alt-1] == '1')  nk\_bit[alt-1] = '0'; // indexing starts from 0  else if (nk\_bit[alt-1] == '0')  nk\_bit[alt-1] = '1';  printf("%d\n", n);  printf("%d\n", k);  printf("%d\n", alt);  printf("%s\n", nk\_bit);  free(nk\_bit);  return 0;  } |

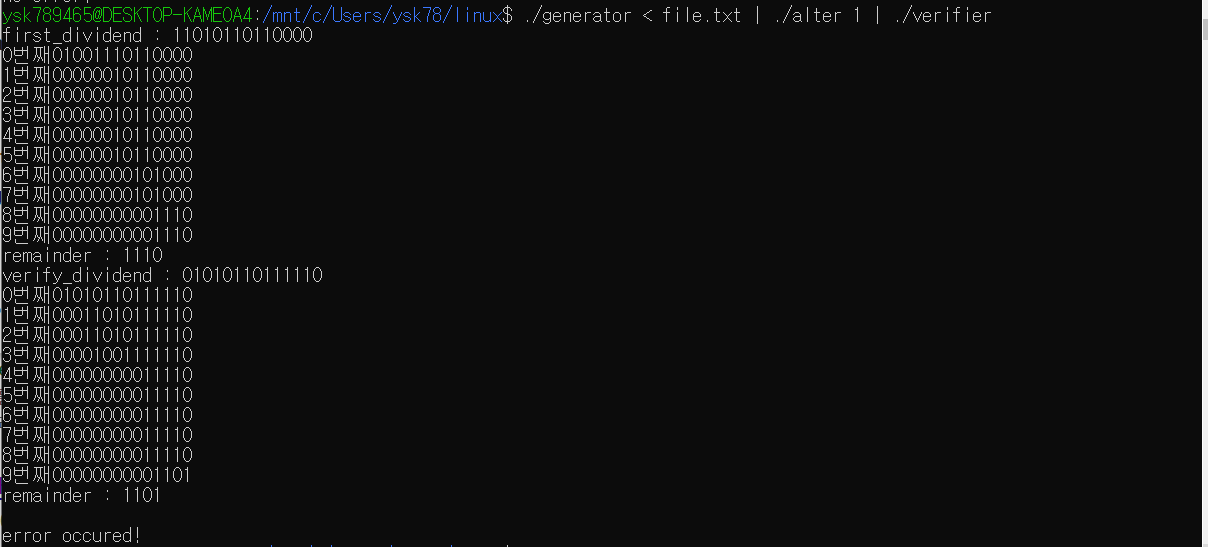
3. verifier 프로그램에서는 generator에서 바로 받거나 alter을 통해 한 비트가 반전된 비트 sequence를 받고, original sequence를 divide 함수를 통해 만든 remainder을 뒤에 붙인 뒤에 다시 divide 함수를 거쳐서 오류를 파악한다.

|  |
| --- |
| #include <stdio.h> // verifier.c  #include <string.h>  #include <stdlib.h>  char\* divide(char\* arr, char\* di, int n, int k);  int main() {  int n, k, alt;  char \*nk\_bit;  char \*divisor; // will divide something  char \*first\_dividend; // will be divided first  char \*verify\_dividend; // will be divided later  char \*remainder;  int i, count = 0;  scanf("%d", &n);  scanf("%d", &k);  scanf("%d", &alt);  nk\_bit = (char\*)malloc(n+k);  scanf("%s", nk\_bit);  divisor = (char\*)malloc(k);  first\_dividend = (char\*)malloc(n+k-1);  verify\_dividend = (char\*)malloc(n+k-1);  remainder = (char\*)malloc(k-1);  for (i = 0; i < k; i++) // to extract k\_bit line  divisor[i] = nk\_bit[i + n];  for (i = 0; i < n; i++) // make first dividend  first\_dividend[i] = nk\_bit[i];  for (i = 0; i < k-1; i++)  first\_dividend[i+n] = '0';  if(alt > 0){ // re-alter to make original bit  if (first\_dividend[alt-1] == '1')  first\_dividend[alt-1] = '0'; // indexing starts from 0  else if (first\_dividend[alt-1] == '0')  first\_dividend[alt-1] = '1';  }  printf("first\_dividend : %s", first\_dividend);  strcpy(remainder, divide(first\_dividend, divisor, n, k)); // first divide to make remainder    printf("remainder : %s\n", remainder);  for (i = 0; i < n; i++) // make verify dividend  verify\_dividend[i] = nk\_bit[i];  for (i = 0; i < k; i++)  verify\_dividend[i+n] = remainder[i];  printf("verify\_dividend : %s", verify\_dividend);  strcpy(remainder, divide(verify\_dividend, divisor, n, k)); // verifying    printf("remainder : %s\n", remainder);  for(i=0; i<k-1; i++){  if(remainder[i] == '1')  count++;  }  if(count >0)  printf("\nerror occurred!\n");  else  printf("\nno error!\n");  free(nk\_bit);  free(divisor);  free(first\_dividend);  free(verify\_dividend);  free(remainder);  return 0;  }  char\* divide(char\* arr, char\* di, int n, int k) {  int i, j, count;  int num\_of\_dividing = n;  char \*tmp\_dividend = (char\*)malloc(n+k-1);  char \*remainder = (char\*)malloc(k-1);  for (i = 0; i < num\_of\_dividing; i++) {  for (j = 0; j < k; j++)  tmp\_dividend[j] = arr[i + j]; // take k bits  if (tmp\_dividend[0] == '0') { // if divisor is bigger than dividend -> divisor \* 0  count = 0;  while (count < k) {  arr[count + i] = ((arr[count + i]-'0') ^ 0) +'0';  count++;  }  }  else if (tmp\_dividend[0] == '1'){ // if divisor is smaller than dividend -> divisor \* 1  count = 0;  while (count < k) {  arr[count + i] = ((arr[count + i]-'0') ^ (di[count]-'0'))+'0';  count++;  }  }  printf("\n%d번째", i);  for (j = 0; j < n+k-1; j++) // set remainder  printf("%c",arr[j]);    }  printf("\n");    for (i = 0; i < k - 1; i++) // set remainder  remainder[i] = arr[i + n];    return remainder;  } |

Alter을 거치지 않고 정상적으로 수행되었을 때의 결과. 확인하기 편하도록 dividend와 나눗셈 과정, 나머지를 print로 출력.



1번째 비트를 alter했을 때의 결과. Error occurred로 오류 표시



3번째 비트를 alter했을 때의 결과. 역시 error occurred로 오류 표시

