# 3.4 출력 생성하기

Monday, October 16, 2023 6:17 PM

## 3.4 출력 생성하기

- 데이터 구조가 생성된 다음에 할 일은 출력을 만들어 내는 것이다.
- 기본 아이디어 : 접두사에 속한 접미사 중 하나를 임의로 선택한 다음에 출력하고, 그 다음 접두사를 처리
- 컨셉 : 접두사를 구성한 단어들을 기억하고 있다가 그 단어로 시작하여, 끝낼 때는 임의로 쓸 단어를 지정해 놓으면 시작과 끝을 구성할 수 있다.

추가 적으로 생각해야 할 점

- 알고리즘을 수행하기에 충분한 입력이 없는 경우에 대한 것
  - 해결방안 1 : 입력이 충분하지 않을 경우 프로그램을 끝내기
  - 해결방안 2 : 신경 쓸 필요가 없도록 입력을 항상 충분히 만드는 것, 우리가 접근할 방식은 이 방식
    - 가짜 접두사를 생성해서 밑작업을 해두면 항상 필요한 입력이 충분하다고 보장할 수 있다.

https://www.cs.princeton.edu/~bwk/tpop.webpage/markov.c

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include "eprintf.h"
```

왜 정의되지 못하고 에러가 나는것인가???

```
enum {
    NPREF   = 2,              /* number of prefix words */
    NHASH   = 4093,           /* size of state hash table array */
    MAXGEN  = 10000           /* maximum words generated */
};
typedef struct State State;
typedef struct Suffix Suffix;
struct State {  /* prefix + suffix list */
    char    *pref[NPREF];   /* prefix words */
    Suffix  *suf;           /* list of suffixes */
    State   *next;          /* next in hash table */
};
struct Suffix {             /* list of suffixes */
    char    *word;          /* suffix */
    Suffix  *next;          /* next in list of suffixes */
};
State   *lookup(char *prefix[], int create);
void    build(char *prefix[], FILE*);
void    generate(int nwords);
void    add(char *prefix[], char *word);
State   *statetab[NHASH];   /* hash table of states */
```

```
char NONWORD[] = "\n";  /* 일반 단어로 사용하지 못하는 단어 */
```

반복문을 시작하기 전에 접두사 배열을 모두 NONWORD로 초기화해 두면, 입력의 첫 번째 단어가 항상 가짜 접두사의 첫 번째 접미사가 된다. 이 방식을 사용하면 반복문에서는 직접 생성하는 접미사만 출력하면 된다.

NONWORD와 같은 존재하지 않는 특별한 단어를 사용하면 코드가 생성 프로세스를 제어하고 시작과 끝이 명확한 텍스트 세그먼트를 생성하여 출력을 보다 관리하기 쉽고 일관되게 만들 수 있다.

```c
/* markov main: 마르코프 체인을 이용한 텍스트 생성 */
int main(void)
{
    int i, nwords = MAXGEN;
    char *prefix[NPREF];            /* 현재 입력 접두사 */
    int c;
    long seed;
    setprogname("markov");
    seed = time(NULL);
    srand(seed);
    for (i = 0; i < NPREF; i++)     /* 최초 접두사를 설정한다*/
        prefix[i] = NONWORD;

    build(prefix, stdin);
    add(prefix, NONWORD);
    generate(nwords);
    return 0;
}
```

앞 장에서 다뤘던 내용들을 모두 종합하여 표준 입력을 읽어서 (지정한 개수 이하로) 단어들을 생성하는 Main 루틴

NONWORD는 일반 입력에서 절대로 나올 수 없는 값이어야 한다. 입력 단어들이 공백 문자로 구분된다는 것을 생각하면, 새줄문자처럼 공백 문자로 구성된 단어를 쓰면 된다.

```c
const int MULTIPLIER = 31;  /* for hash() */
/* hash: compute hash value for array of NPREF strings */
unsigned int hash(char *s[NPREF])
{
    unsigned int h;
    unsigned char *p;
    int i;
    h = 0;
    for (i = 0; i < NPREF; i++)
        for (p = (unsigned char *) s[i]; *p != '\0'; p++)
            h = MULTIPLIER * h + *p;
    return h % NHASH;
}


/* lookup: search for prefix; create if requested. */
/*  returns pointer if present or created; NULL if not. */
/*  creation doesn't strdup so strings mustn't change later. */
State* lookup(char *prefix[NPREF], int create)
{
    int i, h;
    State *sp;
    h = hash(prefix);
    for (sp = statetab[h]; sp != NULL; sp = sp->next) {
        for (i = 0; i < NPREF; i++)
            if (strcmp(prefix[i], sp->pref[i]) != 0)
                break;
        if (i == NPREF)     /* found it */
            return sp;
    }
    if (create) {
        sp = (State *) emalloc(sizeof(State));
        for (i = 0; i < NPREF; i++)
            sp->pref[i] = prefix[i];
        sp->suf = NULL;
        sp->next = statetab[h];
        statetab[h] = sp;
    }
    return sp;
}


/* addsuffix: add to state. suffix must not change later */
```

```c
void addsuffix(State *sp, char *suffix)
{
    Suffix *suf;
    suf = (Suffix *) emalloc(sizeof(Suffix));
    suf->word = suffix;
    suf->next = sp->suf;
    sp->suf = suf;
}


/* add: add word to suffix list, update prefix */
void add(char *prefix[NPREF], char *suffix)
{
    State *sp;
    sp = lookup(prefix, 1);  /* create if not found */
    addsuffix(sp, suffix);
    /* move the words down the prefix */
    memmove(prefix, prefix+1, (NPREF-1)*sizeof(prefix[0]));
    prefix[NPREF-1] = suffix;
}


/* build: read input, build prefix table */
void build(char *prefix[NPREF], FILE *f)
{
    char buf[100], fmt[10];
    /* create a format string; %s could overflow buf */
    sprintf(fmt, "%%%ds", sizeof(buf)-1);
    while (fscanf(f, fmt, buf) != EOF)
        add(prefix, estrdup(buf));
}
```

```c
/* generate: 한 줄에 한 단어씩 출력 생성*/

void generate(int nwords)
{
    State *sp;
    Suffix *suf;
    char *prefix[NPREF], *w;
    int i, nmatch;

    for (i = 0; i < NPREF; i++)        /* 1)접두사 초기화 */
        prefix[i] = NONWORD;

    for (i = 0; i < nwords; i++)       /* 2)텍스트 생성 루프 */
{
        sp = lookup(prefix, 0);        /* 3)접두사 검색 */
        if (sp == NULL)
            eprintf("internal error: lookup failed");

        nmatch = 0;
    for (suf = sp->suf; suf != NULL; suf = suf->next)
        if (rand() % ++nmatch == 0)     /* 4)확률 = 1/nmatch */
                w = suf->word;

        if (nmatch == 0)
            eprintf("internal error: no suffix %d %s", i, prefix[0]);

       if (strcmp(w, NONWORD) == 0)   /* 5)오류처리 */
            break;    /* 6)알고리즘 종료 */

        printf("%s\n", w);

        memmove(prefix, prefix+1, (NPREF-1)*sizeof(prefix[0]));
        prefix[NPREF-1] = w;
    }
}
```

3.4장 앞에서 설명했던 알고리즘을 그대로 구현한 것이다.
이 함수는 한 줄당 한 단어를 출력하기 때문에 9장에서 여러 줄을 한 줄로 합치는 포맷 프로그램 fmt를 다룰 때 자세한 설명 예정

4) 변수 nmatch는 현재까지 일치한 항목의 개수를 세는 변수다. nmatch를 증가시킨 다음, 1/nmatch의 확률로 참이 된다. 따라서 처음 일치한 항목은 확률 1로 선택된 것이고, 두 번째는 1/2의 확률로, 세 번째는 1/3의 확률로 선택된 것이다. 일반적으로 k번째 선택된 항목은 확률 1/k로 선택된 것이다.

2)반복문은 Lookup을 불러서 현재 접두사에 대응하는 해시 테이블 항목을 가져온 다음에 접미사를 하나 선택하고, 선택된 접미사를 출력한 다음에 prefix를 앞으로 당긴다.

5)접미사를 찾을 수 없는 경우 (nmatch=0인경우) 오류를 발생시키며 현재 상태와 접두사의 첫 번째 단어에 대한 정보를 제공 (내부오류 : 접미사 없음)

| | |
|---|---|
| | 6)알고리즘 종료<br>- 선택된 접미사가 NONWORD 경우, 입력 데이터의 끝을 나타내므로 알고리즘 끝<br>- 접미사가 NONWORD 아닌 경우, 접미사를 출력하고 *memmove를 호출해서 접두사에서 첫 번째 단어를 제거한 다음, 선택된 접미사를 접두사의 마지막 단어로 추가하고, 다시 이 과정을 반복 |

*보초값 기법 : 반복 처리를 끝내기 위해, 항상 끝나는 시점을 계산하는대신 특정 값 (보초 값, sential value)을 입력 마지막에 추가한 다음, 이 입력이 나올 경우 반복 처리를 중단하는 방법

*memmove : 일반적으로 메모리 블록을 이동하거나 복사해야 하며 소스와 대상 영역이 서로 겹칠 가능성을 보장할 수 없는 경우 사용된다. 이 함수는 데이터 무결성과 정확성을 보장하며, 메모리 영역이 교차할 수 있는 상황, 예를 들어 원형 버퍼나 다른 동적 메모리 관리 시나리오에서 적합하다.

*간략한 코드 설명
- 마르코프 체인 텍스트 생성기를 구성한 코드
- 기본 아이디어 : 주어진 접두사에 속한 접미사 중 하나를 임의로 선택한 다음에 출력하고, 그 다음 접두사를 처리하도록 하는 것

*6개의 헤더 파일
*헤더 파일 - 개발자가 쉽게 코딩을 하도록 함수나 클래스를 미리 지정해놓은 파일

stdio.h : 입출력 함수
stdlib.h : 탐색과 정렬 함수
string.h :문자열 조작 함수
time.h : 시간 함수
eprintf.h : 사용자가 원하는 것을 구현하기 위해 만든 함수

## 연습문제

연습 문제 3-1. 길이를 모르는 리스트에서 임의의 원소를 선택하는 알고리즘은 난수 생성기 (random number generator)가 좋아야 제대로 작동한다. 어떻게 해야 실제로 잘 돌아가는 난수 생성기를 만들 수 있을 지 설계하고 실험해 보자.

연습 문제 3-2. 각 입력 단어를 별도의 해시 테이블에 저장한다면, 단어가 단 한번만 저장되므로 메모리를 아낄 수 있다. 몇몇 문서를 입력 예로 삼어 얼마나 아낄 수 있는지 측정해보라. 프로그램을 이런 방식으로 구현하면, 해시 체인에서 접두사를 찾을 때에도 문자열 비교 대신 포인터 비교만 하면 되므로, 실행 속도도 훨씬 빨라질 것이다. 이 방식을 써서 프로그램을 구현해 보고 속도와 메모리 소비량에서 얼마나 변화가 있었는지 측정해 보라.

연습 문제 3-3. 데이터의 시작과 끝에 경계표시인 NONWORD들을 추가하는 문장을 제거하고, generate를 고쳐서 경계표시 없이 올바로 시작하고 끝날 수 있게 구현해 보라. 특히 입력의 길이가 0,1,2,3,4,5 단어일 때에도 제대로 동작하도록 만들어라. 그리고 이렇게 만든 코드와 경계 표시를 쓴 코드를 비교해 보라.

의미
prefix : 접두사
surfix : 접미사