**과목명: 시스템프로그래밍**

**1분반**

**<<Project #1>>**

**서강대학교 [컴퓨터공학과]**

**[20161571]**

**[김도연]**

목 차

1. **프로그램 개요**
2. **프로그램 설명**
   1. 프로그램 흐름도
3. **모듈 정의**
   1. 정의한 모듈에 대해 간략적인 설명
4. **전역 변수 정의**
5. **코드 설명**

**6. 보충 가능한 점**

**1. 프로그램 개요**

본 프로그램은 시스템 프로그래밍의 첫번째 프로젝트로 크게 3가지를 구현하는 것을 목표로 한다.

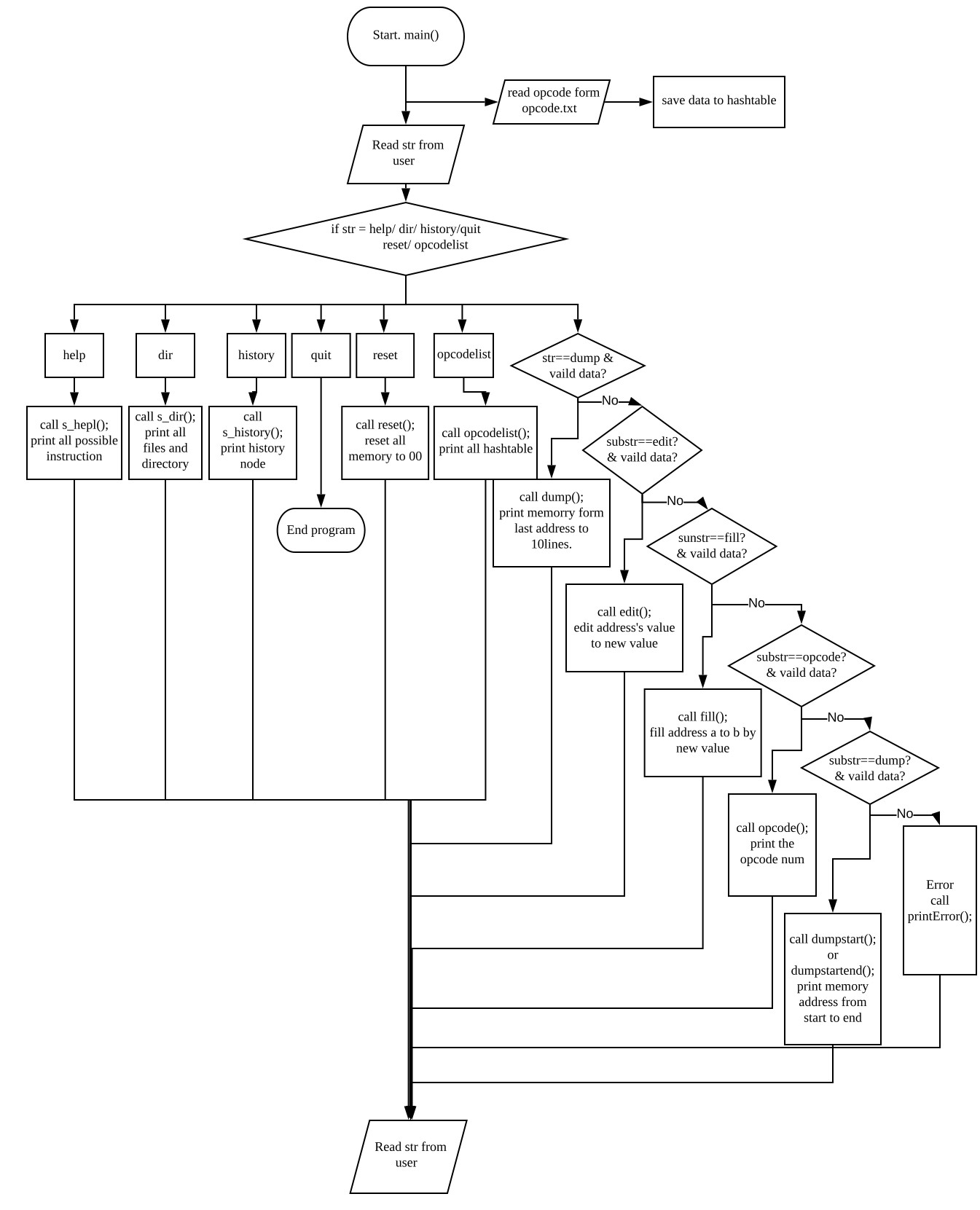
1. UNIX shell처럼 동작하는 sicsim과 그 명령어들(help, dir, quit, history)

2. 1MB메모리 공간과 그 메모리 관련 명령어들(dump…, edit…, fill…, reset)

3. opcode를 담을 해시테이블과 관련 명령어들(opcode …, opcodelist)

기본적으로 이 프로그램은 각 명령에 따라 각각의 동작을 수행하는데 모든 명령은 sicsim> 에서 user에게 명령어를 입력 받아 이루어지며, 입력 받은 명령이 유효한 명령일 경우 history에 저장된다. quit가 입력되면 프로그램이 종료되고, 그외의 모든 명령어는 수행이 끝나면 다시 sicsim> 화면으로 돌아와서 user의 입력을 기다린다. 본 프로그램을 위해서는 history를 위한 Linkedlist 자료구조. 그리고 opcode명령어를 위한 해시테이블 구조를 구현해야 한다. 자칫 코드가 길어질 수 있다 보니, 반복적인 작업은 최대한 함수로 처리할 수 있게 구현하려고 노력하였다.

**2. 프로그램 설명**

2-1. 프로그램 흐름도

**3. 모듈 정의**

3.1 정의한 모듈에 대해 간략적인 설명

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*shell 관련 모듈\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. void s\_help(void)

가능한 모든 명령어들을 출력해주는 함수  
2. void s\_dir(void)

현재 디렉토리에서의 디렉토리들과 파일들을 출력해주는 함수

<sys/stat.h> 와 <dirent.h>를 이용해서 구현하였다.

3. void s\_history(List \*)

현재까지 실행된 유효한 명령어들을 차례대로 출력해주는 함수.

linked list로 연결되어 있는 List\* 구조체를 매개변수로 받아서 처음부터 끝까지 출력해준다

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Memory 관련 모듈\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

4. Node\* makeNode(char\*)

char\*에 저장되어 있는 문자열을 data로 갖는 Node를 하나 동적 할당하고 Node->next=NULL로 초기화 한다. 그리고 그 Node의 주소 값인 Node\*를 return 한다.

5. void Init(List\*, char\*)

인자로 전달받은 List 구조체를 초기화한다. Linkedlist 구조를 위한 dummy노드를 만들고 초기화한다.

6. void InsertNode(List\* , char\*)

전달받은 str을 data로 갖는 Node를 하나 만들고 전달받은 List의 Linkedlist의 맨 끝에 삽입한다.

7. void Dump(int\*)

전달받은 int\*의 address 부터 memory를 10line 출력한다. 다음 시작주소를 int\*에 다시 저장한다. 전달받은 값에 대한 에러체크 또한 수행한다.

8. void DumpStart(int, int\*)

전달받은 int 주소의 address 부터 memory를 10line 출력한다. 다음 출력할 주소를 int\*에 저장한다.전달받은 값에 대한 에러체크 또한 수행한다.

9. void DumpStartEnd(int, int, int\*)

전달받은 int 주소의 address 부터 전달받은 int 주소의 address까지 memory를 출력한다. 범위에 대한 에러체크 또한 수행하며 다음 출력할 주소를 int\*에 저장한다.

10. void EditAddress(int, int)

전달받은 int 주소의 값을 전달받은 int 값으로 변경한다. 값의 범위에 대한 에러체크 또한 수행한다.

11. void Fillvalue(int, int, int)

전달받은 int 주소부터 전달받은 int 주소까지 전달받은 int 값으로 변경한다. 값의 범위에 대한 에러체크 또한 수행한다.

12. void reset(void)

모든 memory의 값을 00으로 초기화 한다.

13. void printArr(int, int)

위의 모든 메모리 출력동작에 사용되는 함수이다. 전달받은 int 값부터, 전달받은 int값까지 서식에 맞춰 출력해준다.

14. void printError(void)

잘못된 명령어에 대한 Error 출력함수이다.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Hash 관련 모듈\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

15. HNode\* makenewnode()

node를 하나 동적 할당하고 모든 구성요소(key,opnum,opArr,dnum,->next)를 초기화한다. 그리고 그 주소인 HNode\*를 return한다.

16. int getKey(char\*)

Key값을 구하는 해시 함수이다. 전달받은 char\* 문자열을 다 더한 후 mod20연산을 통해 int key값을 구하고 그 값을 return 한다.

17. void opcodelist()

해시테이블을 모두출력해주는 함수이다. 전역변수로 선언한 hpoint로 hArry[i]를 가리키

고 hpoint가 가리키는 게 NULL이 될 때까지 이동하며 출력한다. 한 줄을 끝내면 다음 index로 이동하여 반복한다.

18. void SearchCode(char\* str)

전달받은 char\* 문자열을 통해 key를 구하고 해시함수를 탐색하여 해당하는 opcode의 숫자를 출력해준다.

19. void InitHash()

포인터 배열을 모두 NULL로 초기화한다.

**4. 전역 변수 정의**

프로그램의 용이한 수정을 위해 전역변수 사용을 최소화하였다. 그럼에도 이 프로그램에서 정의한 전역변수는 모두 3가지이다.

1. 1MB 메모리 공간을 위한 unsigned char mArr[65536][16]

프로그램 특성상 mArr을 가지고 하는 연산이 굉장히 많은데, 그때그때 매개변수로 모두 넘겨주다 보면 코드가 복잡해지고 수정하기도 불편할 것 같아 전역배열로 선언하였다.

2. 해시테이블을 위한 HNode\* hArry[20] 포인터 배열

위와 마찬가지로 해시 포인터 배열 역시 그때그때 넘기기 보다는 프로그램의 간결성을 위해 전역 포인터 배열로 선언하였다.

3. 그 포인터 배열의 각 노드를 가리키기 위한 HNode\* hpoint 포인터

1,2번과 같은 이유이다.

**5. 코드 설명**

처음 프로젝트 명세서를 봤을 때, 핵심은 LinkedList와 해시 테이블이라고 생각했고 생각보다 매우 수월하게 코딩 할 수 있을 것 같았다. 가장 처음에 구현한 기능은 UNIX shell처럼 동작하게 하는 shell 관련 함수였다. help는 그냥 printf 출력 기능이고, history의 경우 Linkedlist 구현이 필요한데, 가장 기본적인 방식으로 선언하였다.



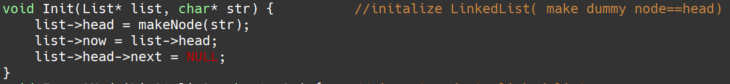
<명령어를 저장할 node 구조체>

그리고 이번 프로젝트에서는 Linkedlist 연산 중 리스트의 출력/ 삽입만 필요하기 때문에 훨씬 더 간단하게 프로그래밍 할 수 있었다.



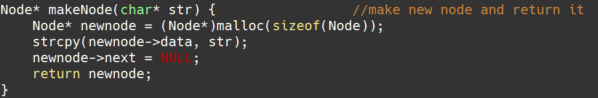
<Linkedlist 구현을 위한 List 구조체>

또한 dummy Node를 구현하여 Node 삽입시 한가지 방법으로 할 수 있게 하였다.

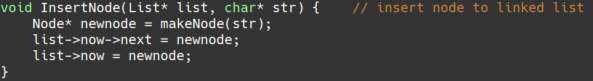


<List 초기화>

새로운 노드의 생성/ 삽입 역시 따로 각각 구현해서 main에서 InsertNode 함수를 호출하고, InsertNode함수에서 makeNode함수를 호출하게끔 구현했는데, 최대한 main함수를 깔끔하게 하려고 노력하였다.

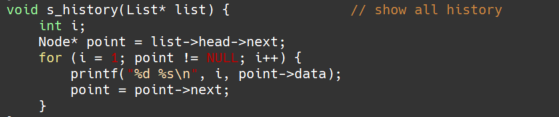


<Node생성후 값 할당>.



<노드 삽입 함수>

마지막으로 history함수에서는 List의 처음부터 끝까지 출력하게 하였다. for문과 Node\* pointer를 이용하였다.



<history 함수>

마지막으로 dir 함수를 구현하였는데, <sys/stat.h>와 <dirent.h> 해더를 추가해서 선언되어 있는 구조체, 매크로를 이용하였다. 구현해야 하는 기능은, 현재 디렉토리 내에 있는 모든 파일과 디렉토리 출력, 그리고 파일 중 실행파일은 \*를 붙여서 출력이다.

dir 함수에서 고려한 점들은

1. Linux상에서 system call로 ls했을때와 같은 결과를 출력해야 하기 때문에 숨김 파일은 출력하지 말아야 한다.

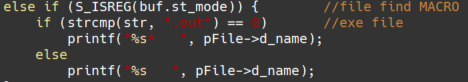


struct dirent\*의 맴버인 d\_name에 파일/디렉토리들을 저장했는데, 숨김 파일은 .으로 시작이기때문에 숨김 파일이 아닐 때에만 출력함수에 진입하게 하였다.

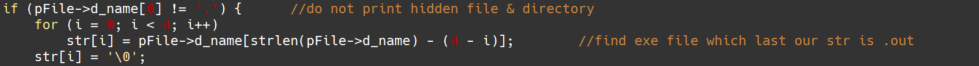
2. 파일과 디렉토리의 구분, 그리고 파일과 실행파일의 구분



해더에서 제공하는 매크로인 S\_ISDIR(buf.st\_mode)를 사용해서 directory를 선별했고



S\_ISREG(buf.st\_mode) 매크로를 사용해서 파일임을 확인하였다. 그리고 실행파일의 구분은 실행파일은 .out이라고 가정하여, d\_name의 마지막 4문자를 확인하여 .out이면 실행파일, 아니면 일반파일로 분류하였다.



이렇게 shell 부분의 구현은 어렵지 않게 끝났다. 그리고 memory파트를 구현하기 전에, 이미 함수의 프로토타입은 다 선언해 놨기에, 메인 함수부터 다 디자인하고 마지막에 나머지 함수들을 채우려고 하였다. 메인에서는 string의 입력 그리고 그에 따른 분기만 나눠주면 되었기에 매우 간단할 줄 알았는데, 여기서 문제가 발생했다. help,dir,hi,reset 등등 이런 명령은 아무런 문제가 되지 않았는데 문제는 dump쪽이 였다. dump, dump x, dump x,y를 구현해야 하는데, 어떤 방식으로 구현할지 꽤 오래 고민을 했다. scanf로 따로따로 받고 그에 따른 분기를 설정할지, 아니면 한번에 문자열을 통째로 받아서 나눌지. 나는 결국 두번째 방법을 택했는데, 잘한 선택인지는 모르겠다. 이 때문에 main 함수가 상당히 길어졌다.

main의 디자인은 다음과 같다.

우선 fgets로 문자열을 통째로 받는다.



먼저 strcmp를 이용하여 str와 단일단어 명령(help,quit,history,reset,opcodelist,dump)를 처리한다. 각각의 분기에서는 명령어를 history Linkedlist에 삽입하고 각 함수로 이동하여 디자인된 동작을 수행한다. 문제는 하나이상의 단어가 필요한 명령어들 이였는데, 각 문자열을 보니 각각의 인자가 ‘ ’ 또는 ‘,’로 나눠져 있었다. 그래서 substr이라는 서브 스트링을 하나 선언해서 인자들을 하나 하나씩 담으면서 판단하기로 하였다.



, 나 ‘ ’전까지 str를 substr에 저장한다. 이는 첫번째 인자를 알아내기 위한 방법이다. 첫번째 인자를 알아냈으면, 여기서 또 각분기로 갈라진다.









첫번째 인자가 가질 수 있는 단어의 종류는 총 4가지이며, 그 외의 단어는 모두 invalid cmd로 Error처리한다.

두번째 인자는 ‘ ’나 ‘,’뒤에 시작하므로 i+1만큼땡겨서 str[i]를 선택해주고, j는 다시 0으로 초기화해서 substr은 다시 두번째 인자를 저장할 수 있게 해주었다.





그리고 각각의 경우에 적절한 Error처리를 해주었다.

ex) 인자가 부족한 경우, 인자로 이상한 게 오는 경우

3번째인자가 필요한 dump [start, end],edit, fill 명령어의 경우 위의 식을 한번씩 더 반복하면 똑 같은 결과를 얻을 수 있다. 문제는 문자열에 저장된 알파벳,숫자문자를 16진수로 인식하는 방법이었는데, 좀 찾아보다보니 방법이 있었다.



strtol 함수는 첫번째 인자로 문자열, 두번째 인자로는 문자열이 아닐 경우 아닌 위치를 2번째 포인터에 저장, 3번째 인자로는 진법을 가진다. 이 함수로 문자열의 char를 바로 16진수로 바꿀 수 있게 되었다. 이로써 모든 분기들의 경우는 다 완료하였고 dump 계열 명령어는 마지막 출력주소의 위치를 기억해야 할 필요가 있었기에



를 통해서 memory address를 기억하게끔 하였다.

이제 memory관련함수를 추가로 설정하자. 전달되는 인자의 범위 에러체크는 기본으로 하고 Dump, DumpStart, DumpStartEnd 함수는 모두 memory를 출력하는 함수이기에 printArr라는 하나의 함수를 설정해서 전달되는 인자만 다르게 하면 훨씬 더 편하게 출력할 수 있을 거 같았다.

printArr함수는 기본적으로 start와 end값을 전달받아서 출력한다.

이때 각 함수에 맞게 설정해서 넘겨주면 되는데 Dump의 경우는 10line 출력임으로 아래와 같이,



DumpStart의 경우는 시작 주소 +10line 출력임으로 아래와 같이



마지막으로 DumpStartEnd의 경우는 start, end주소를 넘겨주게 만들었다.



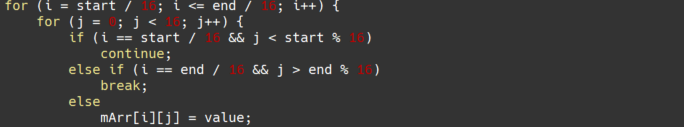
또 세 경우모두 마지막 주소를 저장해야 하기에 각 함수가 종료되기전 \*point에 마지막 주소를 저장해 주었다.

EditAddress와 Fillvalue는 위 Dump류 함수와는 다르게 자체적으로 함수 내부에서 처리하였다.

EditAddress의 경우 주어진 주소에 해당 값을 넣으면 되는 함수기에 address를 전달받아 행렬만 구하면 되었다. 16열이고 16진수임으로 아래와 같은 식을 통해 완성하였다.



Fillvalue역시 start와 end주소를 받아서 시작과 끝행을 구해서 그 주소들을 해당 값으로 채워 넣었다. 코드는 다음과 같다.



reset 함수의 경우 memset을 통해 memory배열 전체를 0으로 초기화 하였다.



마지막으로 printArr 함수는 위의 Fillvalue함수와 비슷하다. 시작주소와 끝주소를 전달받아 가운데 memoryArr를 출력할 때는, 빈칸도 출력해야 하기에 첫 행과 마지막행일때는 따로 예외 처리를 하였다.



<첫 행일 때 빈칸 채워야 할 경우>.



<마지막 행일 때 빈칸 채워야 할 경우>

그리고 가장 왼쪽의 주소 값은 첫번째 for문이 실행될 때마다, 즉 메모리의 행이 바뀔 때마다 가장 앞에 찍어 주었고, 오른쪽의 ASCII값은 그냥 해당 값을 char형식으로 출력해 주었다.

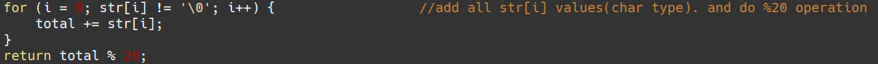
마지막으로 해시테이블을 구현할 때 고려해야할 상황들은

1. key를 결정할 해시함수

2. 중복되는 key를 처리할 방법.

3. 해시테이블을 구현할 자료구조

해시함수는 opcode의 문자열부분을 모두더한값을 mod20연산을 통해 정하기로 하였다.



<GetKey 함수>

중복되는 key값은 체이닝을 이용해서 처리하기로 결정하였다. 그러려면 Linkedlist 구조를 사용해야 했기에, 당연히 해시테이블의 구현 또한 일반배열이 아닌 포인터배열을 이용해서 구현하기로 하였다.

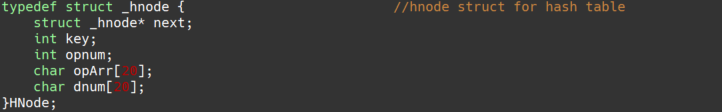
포인터배열은 전역변수로 선언하였다.



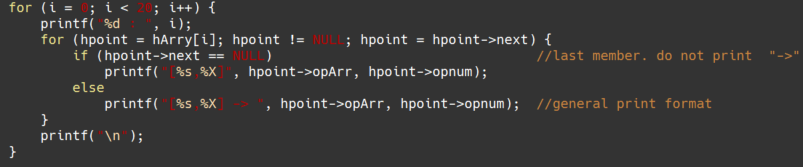
데이터의 저장과 검색, 출력을 위한 포인터도 하나 선언하였다.



또 저장해야 할 data들이 key까지 합해서 총 4개였기에 Node 구조체는 다음과 같이 구성하였다.



opcodelist의 동작인 리스트 전체출력은 아까 history출력을 각 행마다 반복하는 식으로 똑같이 작성하였다. 다만 마지막행은 ->표시가 없어야 하기에 그 부분만 따로 분리하였다.



opcode xxx 함수가 수행하는 동작인 해당 문자열을 찾아서 숫자data를 출력해주는 함수는 Linkedlist의 탐색과 동일하게 작성하였다.

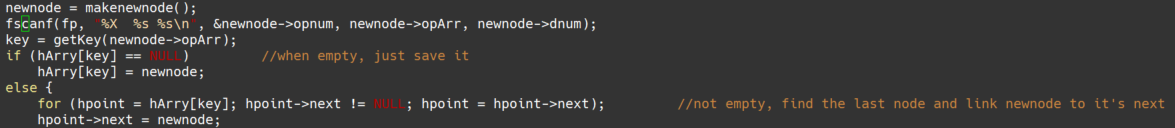
문자열이 입력으로 들어오면 해당 GetKey함수를 통해 key값을 구하고 그 값으로 해당 해시테이블의 행으로 이동하여 일치하는 문자여부를 확인한다.

물론 이모든과정에 앞서 main에서 추가로 opcode.txt.에서 세개의 데이터를 읽어 해시테이블에 저장해야 한다.

파일의 끝에 도달할 때까지 while문을 수행한다.



그 후에 노드를 하나 생성한 후 각각 데이터의 형식에 맞게 (%X %s %s) fscanf로 데이터를 입력 받는다. 이때 명령어인 str은 입력받는즉시 GetKey함수를 이용해 key값을 구하고, 해당 노드의 주소를 해시테이블에 바로 저장한다. 해시테이블이 꽉차있다면, 그 행의 제일 끝 노드로 가서 거기에 저장한다.



**6. 보충가능한 점**

프로그램을 완성하고 보니 부족했던 점들이 눈에 보였다. 가장 엉망이었던 것은 문자열을 입력하고 또 처리하는 방식이었다. 한번에 fgets로 문자열 통째로 받고 또 그걸 분리하는 과정은 굉장히 복잡하고, 또 코드의 수정에 있어서 매우 분리하다. 또 여러가지 예외처리에 있어서도 매우 힘들었다. 현 시점에서 생각해본 가장 이상적인 방법은 scanf로 하나하나 따로따로 받는 방식이다. 처음에는 그 후의 처리가 생각이 안나서 포기했는데, 생각해보니 scanf로 입력 받으면 그 입력이 끝났다면 입력 버퍼에 ‘\n’이 남을 것이고 이를 통해 인자가 더 있는지 여부를 확인할 수 있었을 것 같다. 그럼에도 불구하고 이번 프로젝트를 진행하면서 C언어가 문자열을 다루기에 정말 별로인 언어라는 것을 다시한번 깨달았다. 다른 친구들은 어떤 식으로 입력 받았는지 얘기해보면 괜찮을 거 같다는 생각을 했다.

그리고 이 프로젝트에서는 해더파일을 하나로만 나눴는데, sell관련, memory관련, opcode관련으로 각각 하나씩 나눴으면 추후 프로젝트에서 훨씬 더 편하지 않을까 하는 생각이 들었다.