**객체지향 프로그래밍**

**Assignment 1 보고서**

창의IT융합공학과 20130392

정건화 (ghjeong12)

**명예서약(Honor code)**

**“나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.”**

1. **프로그램 기능에 대한 개요**

이번 과제에서는 피자가게 메뉴 관리 및 주문 시스템을 구현했다. 이를 통해 클래스를 이용하는 법을 익혔다. **본 프로그램은 과제 요구사항의 권장에 따라 총 13개의 파일로 이루어져 있다.** 파일의 종류는 아래와 같다.

Topping.cpp Dough.cpp Edge.cpp Topping.h Dough.h Edge.h

ToppingList.cpp DoughList.cpp EdgeList.cpp ToppingList.h DoughList.h EdgeList.h

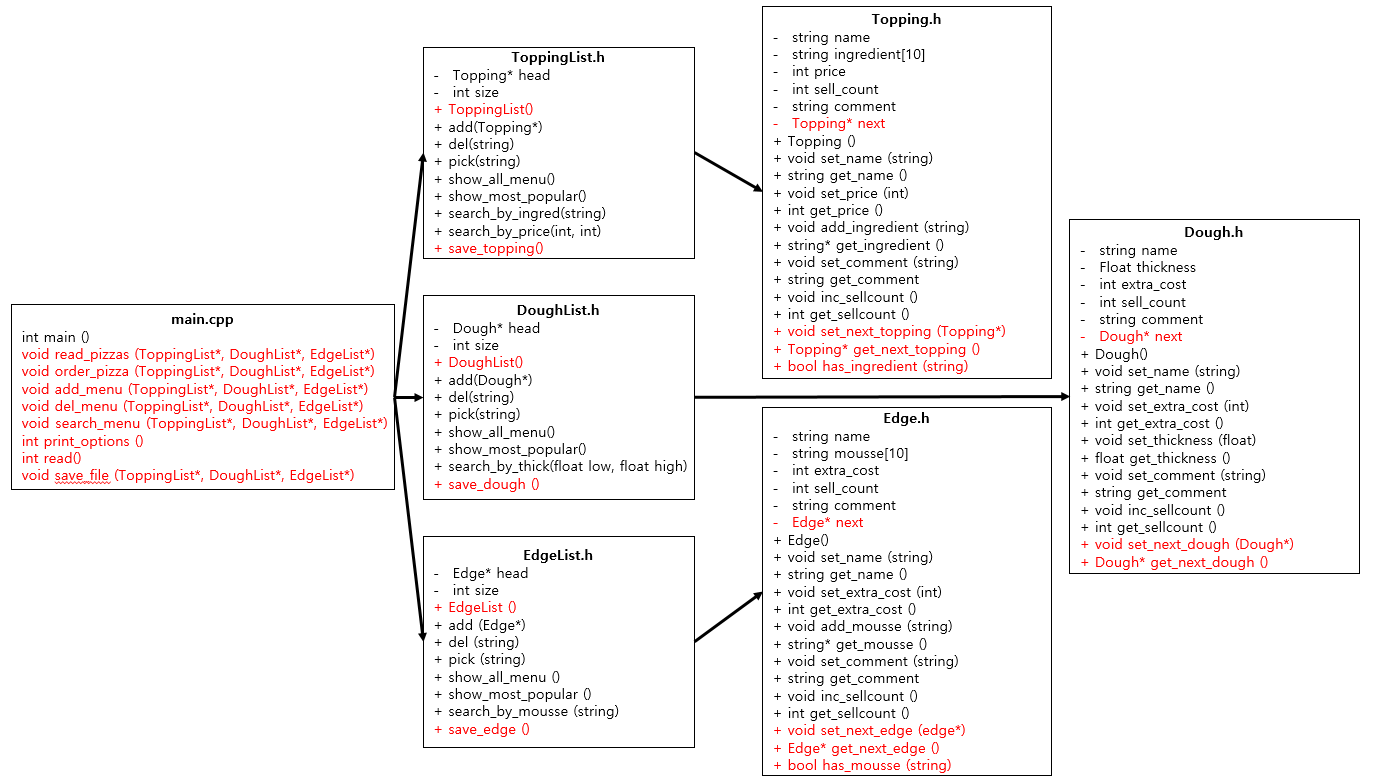
main.cpp

토핑과 도우, 에지 클래스를 관리하기 위해 각각 클래스 별로 cpp 파일과 h 파일을 만들었으며, 이와 관련된 list 클래스를 관리하기 위해 각각 클래스 별로 cpp 파일과 h 파일을 만들었다. **각 메뉴와 리스트는 모두 클래스로 구현하였다.** 따라서 본 프로그램은 총 12개의 클래스와 하나의 main 함수로 이루어진다. **각 클래스의 모든 멤버 변수는 ‘private’ 으로 선언하였고, 이에 대한 접근은 ‘public’ 멤버 함수를 통해서만 이루어지도록 하였다.**

main.cpp 에 정의된 main 함수 안에서 사용자의 입력을 받아 해당 함수를 실행한다. 사용자의 입력에 따라 0은 모든 메뉴 보기, 1은 주문하기, 2는 메뉴 추가, 3은 메뉴 삭제, 4는 메뉴 검색, 5는 인기메뉴 추천, 6은 프로그램을 종료를 수행한다. **7번은 새로 추가한 명령으로, 현재 메뉴를 텍스트 파일로 저장하도록 한다.** 1, 2, 3, 4, 7 의 입력에 해당하는 주문하기, 메뉴 추가, 메뉴 삭제, 메뉴 검색은 코드가 길어 main.cpp 에서 함수로 추출하여 정의하여서 main 함수의 가독성을 높였다. **사용자의 입력이 6 이 아닌 경우, 과제의 요구사항에 따라 반복 실행되도록 하였다. 또한 상속과 STL이나 Boost 등의 라이브러리는 사용하지 않았다.** 프로그램이 사용자의 입력을 받기 전, init\_menu.txt 파일을 읽어서 해당 파일에 있는 토핑과 도우, 에지 정보를 미리 저장한다. init\_menu.txt 파일의 구성은 과제의 프로그램 설명 부분에 나와있으며, 간단히 설명하면 –토핑, -도우, -에지를 이용하여 각 부분을 분리한다.

1. **클래스 구성 및 알고리즘**

본 프로그램은 6개의 클래스와 메인 함수로 이루어진다. 메인 함수에서는 ToppingList.h, DoughList.h, EdgeList.h 를 #include 전처리문을 이용하여 포함시킨다. 또한, ToppingList.h는 Topping.h를, DoughList.h는 Dough.h를, EdgeList.h는 Edge.h 를 #include 전처리문을 이용하여 포함시킨다. 따라서 ToppingList 클래스는 Topping 클래스를, DoughList 클래스는 Dough 클래스를, EdgeList 클래스는 Edge 클래스를 이용한다. 아래 그림에서 빨간색으로 표시된 항목은 새롭게 추가한 함수 혹은 변수다. A에서 B로 가는 화살표는 A에서 B를 #include 를 통해 넣어준 것을 의미한다.

본 과제에서는 과제 문서에 주어진 0~6번 입력 외에 **7번 기능을 추가했다.** 7번을 입력하면 현재 토핑/도우/에지 메뉴를 **save\_menu.txt로 저장하도록 하였다.** 이를 통해 나중에 현재 상태를 save\_menu.txt를 활용하여 불러올 수 있다.

6개 각각의 클래스의 멤버 변수와 함수들에 대해 설명을 하겠다. 먼저 *Topping* 클래스는 토핑의 이름을 저장하는 name, 재료를 저장하는 string의 배열 ingredient, 가격을 저장하는 price, 주문된 횟수를 기억하는 sell\_count, 토핑에 대한 소개인 comment, 마지막으로 **새롭게 추가한 Topping\* next가 있다**. 모든 멤버 변수는 private으로 선언하였다. 멤버 함수로는 토핑 이름을 넣고 얻는 set\_name 과 get\_name 함수가 있으며, 가격을 넣고 얻는 set\_price 와 get\_price 함수가 있다. 재료들의 배열을 얻는 get\_ingredient 와 재료 하나를 배열에 추가하는 add\_ingredient 함수가 있다. 설명을 넣고 얻는 set\_comment 와 get\_comment 함수가 있고, sell\_count를 1 증가시키는 inc\_sellcount 함수와 반환해주는 get\_sellcount 함수가 있다. 마지막으로, 새로 추가한 멤버 함수로는 name, comment, ingredient의 각 원소들을 “”으로 초기화하고, price, sell\_count, next를 0으로 초기화하는 컨스트럭터인 Topping()과 링크드리스트에서 다음 토핑을 저장하는 set\_next\_topping 함수와 다음 토핑의 주소를 반환하는 get\_next\_topping 함수가 있다. 재료 하나의 이름이 주어졌을 때, 이 토핑의 재료에 해당 재료 하나가 들어있는지 확인해서 bool 타입으로 반환하는 has\_ingredient 함수도 만들었다.

다음으로 *ToppingList* 클래스에서는 멤버 변수로 리스트의 시작 노드인 head와 리스트의 크기인 size가 있다. 이는 모두 private으로 선언하였다. 멤버 함수로는 리스트에 Topping\* 를 추가하는 add 함수와 이름이 주어졌을 때 해당 이름에 해당하는 node를 삭제하는 del 함수가 있다. del 함수는 노드 삭제에 성공하면 true, 실패하면 false를 반환한다. 또한 이름이 주어졌을 때 리스트에서 해당 이름에 해당하는 토핑 객체의 주소를 반환하는 pick 함수와 모든 메뉴를 보여주는 show\_all\_menu, 가장 많이 주문한 토핑을 보여주는 show\_most\_popular 함수가 있다. search\_by\_ingred 함수는 재료 이름을 받아서 해당 재료를 갖고 있는 토핑을 모두 보여주고, search\_by\_price 함수는 가격의 하한과 상한을 받아 하한 이상 상한 이하의 가격을 갖는 모든 토핑들을 보여준다. 마지막으로 ToppingList의 파라미터가 없는 컨스트럭터를 **새로운 멤버 함수로 추가**하였다. 이 컨스트럭터는 head와 size를 0으로 초기화한다. 토핑 리스트에 있는 토핑들을 모두 save\_menu.txt에 저장하는 save\_topping 함수도 **새로 추가하였다.**

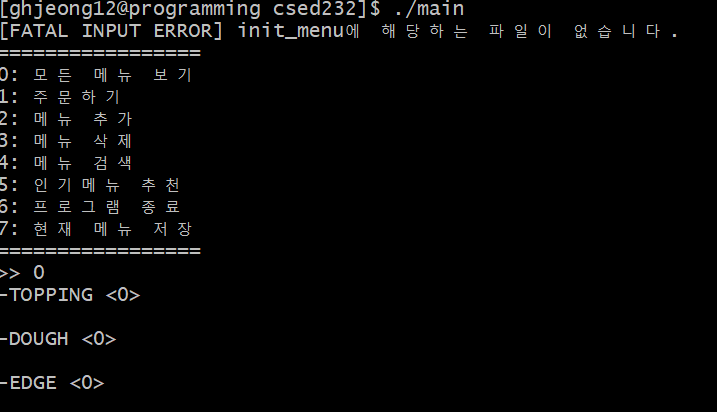
다음으로 *Dough* 클래스를 설명하겠다. Topping 클래스의 함수와 기능이 같은 경우, 중복된 내용이므로 자세히 설명하지 않겠다. Dough 클래스의 멤버 변수로는 이름을 저장하는 name과 굵기를 저장하는 thickness, 추가비용을 저장하는 extra\_cost, 주문된 횟수를 저장하는 sell\_count, 설명을 저장하는 comment, 마지막으로 **새롭게 추가한 멤버 변수 Dough\* next가 있다.** 이들은 모두 private으로 선언했다. next는 링크드리스트를 유지하는데 이용된다. 멤버 함수로는 Topping 클래스와 같은 기능을 하는 set\_name, get\_neame, inc\_sellcount, get\_sellcount, set\_comment, get\_comment가 있다. 이와 더불어, 추가 비용을 넣는 set\_extra\_cost, 이를 얻는 get\_extra\_cost, 굵기를 넣고 얻는 set\_thickness, get\_thickness 함수가 있다. **또한 새로 추가한 멤버 함수로** 다음 Dough 주소를 저장하는 set\_next\_dough와 다음 Dough 주소를 반환하는 get\_next\_dough 함수가 있다. 또한 Dough() 컨스트럭터를 만들어서 name과 comment를 “”으로, extra\_cost, sell\_count, next, thickness를 0으로 초기화한다.

*DoughList* 클래스 역시 ToppingList 클래스와 겹치는 함수는 자세히 설명하지 않겠다. 멤버 변수로는 리스트의 시작 노드인 head와 리스트의 크기를 갖는 size를 private으로 선언했다. 멤버 함수로는 리스트에 노드로 Dough의 주소를 추가하는 add와 도우 이름을 받아서 해당하는 도우를 리스트에서 삭제하는 del 함수가 있다. del 함수는 노드 삭제에 성공하면 true, 실패하면 false를 반환한다. 도우 이름이 주어지면 리스트에서 해당하는 도우를 찾아 그 주소를 반환하는 pick 함수와 모든 메뉴를 보여주는 show\_all\_menu, 가장 많이 주문된 메뉴를 보여주는 show\_most\_popular 함수가 있다. 두께의 상한과 하한을 받아 하한 이상 상한 이하의 도우들을 출력하는 search\_by\_thick 함수가 있으며, **새롭게 추가한 멤버 함수**로 파라미터가 없는 컨스트럭터를 만들었다. 이 컨스트럭터는 head와 size를 0으로 초기화한다. 도우 리스트에 있는 도우들을 모두 save\_menu.txt에 저장하는 save\_dough 함수도 **새로 추가하였다.**

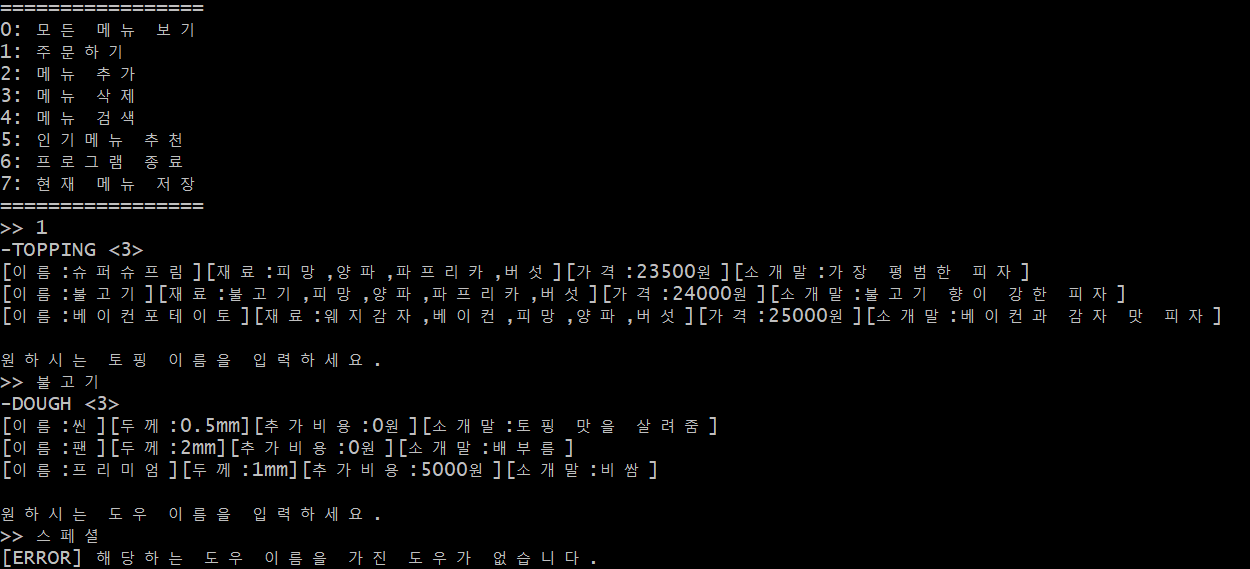
*Edge* 클래스는 private 멤버 변수로 이름을 나타내는 name과 추가비용을 나타내는 extra\_cost, 주문된 횟수를 나타내는 sell\_count, 설명을 나타내는 comment, 무스의 스트링 배열(size=3)인 mousse를 갖는다. 또한 **새롭게 추가한 멤버 변수**로 링크드리스트를 위해 Edge\* 타입의 next를 갖는다. 멤버 함수들로는 토핑/도우 클래스에서와 같은 역할을 하는 set\_name, get\_name, set\_extra\_cost, get\_extra\_cost, inc\_sellcount, get\_sellcount, set\_comment, get\_comment가 있다. 무스 이름을 받아 mousse에 추가하는 add\_mousse함수와 mousse를 반환하는 get\_mousse 함수도 있다. 마지막으로 **새롭게 추가한 멤버 함수로는** 링크드리스트를 지원하기 위해 set\_next\_edge 함수와 get\_next\_edge 함수를 이용해 에지의 주소를 리스트에 next에 저장하고 얻는 함수를 만들었다. 또한 name 과 comment를 “”, extra\_cost, sell\_count, next를 0, mousse의 원소 하나 하나를 “”으로 초기화하는 Edge() 컨스트럭터를 만들었다. 마지막으로 스트링 하나를 받아서 스트링의 배열인 mousse안에 해당 스트링에 해당하는 무스가 들어 있는지 확인하여 bool 타입으로 반환하는 has\_mousse 함수도 만들었다.

마지막으로 *EdgeList* 클래스를 설명하겠다. 먼저 멤버 변수로 리스트의 시작 노드인 head와 리스트의 크기를 갖는 size를 private 으로 선언했다. 멤버 함수로는 Edge의 주소 하나를 리스트에 추가하는 add 함수와 이름을 받아서 해당 이름에 해당하는 에지를 리스트에서 삭제하는 del 함수 (del 함수는 노드 삭제에 성공하면 true, 실패하면 false를 반환한다), 에지 이름을 받아서 리스트에서 이에 해당하는 Edge의 주소를 반환하는 pick 함수, 모든 메뉴를 보여주는 show\_all\_menu, 가장 주문이 많이 된 에지를 보여주는 show\_most\_popular 함수, 그리고 특정 무스 하나를 선택했을 때, 그 무스를 사용하는 에지를 모두 출력하는 search\_by\_mousse 함수가 있다. **새로 추가한 멤버 함수**로는 EdgeList() 로 파라미터가 없는 컨스트럭터가 있다. 이 컨스트럭터는 head와 size를 0으로 초기화한다. 에지 리스트에 있는 에지들을 모두 save\_menu.txt에 저장하는 save\_edge 함수도 **새로 추가하였다.**

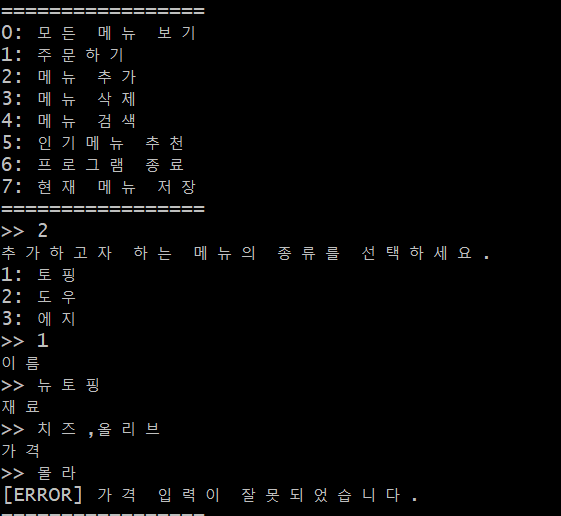
main.cpp 에서 main 함수를 간단하게 하기 위해 몇가지 함수를 추출하여 이를 정의하고 사용하였다. 이들을 간단히 설명하겠다. 먼저 read\_pizzas 함수는 ToppingList\*, DoughList\*, EdgeList\* 를 파라미터로 받는다. 그리고 init\_menu.txt의 파일을 열고 과제 문서에 적힌 형식대로 읽어서 파라미터로 받은 리스트에 토핑, 도우, 에지를 저장한다. 이 때, init\_menu.txt 파일 읽기에 실패하면 아래와 같이 에러 메시지를 출력하고 아무것도 리스트에 저장하지 않고 함수를 마치고 프로그램을 계속 실행한다. 즉, 기존에 저장된 토핑과 도우와 에지가 하나도 없이 프로그램을 실행하게 된다.



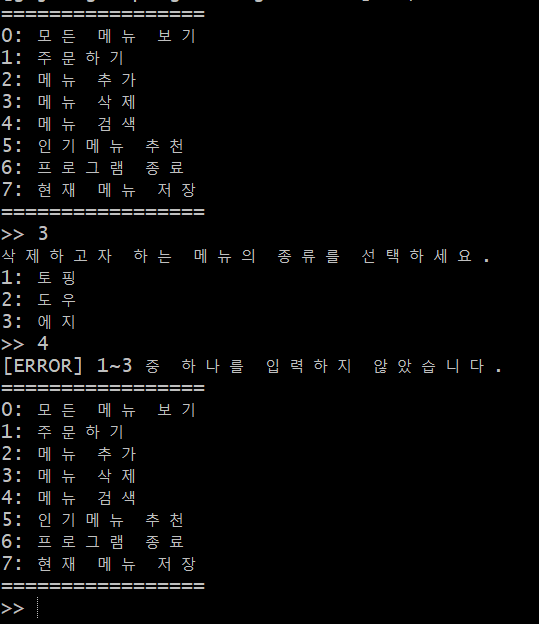
다음으로 order\_pizza 함수는 주문하기에 해당하는 함수로 ToppingList\*, DoughList\*, EdgeList\* 를 파라미터로 받는다. 원하는 토핑 이름과 도우 이름, 에지 이름을 입력 받아서 각각의 이름과 가격을 더해 총 이름과 가격을 출력한다. 이 때, 과제 문서에 주어진 예시에 따라 이름은 “도우 에지 토핑” 순으로 한다. 또한 sell\_count를 inc\_sellcount를 이용해 증가시켜 인기 메뉴에 반영되도록 한다. 만약 해당하는 토핑/도우/에지의 메뉴가 없다면 아래와 같이 에러 메시지를 출력하고 프로그램을 처음부터 다시 실행한다.



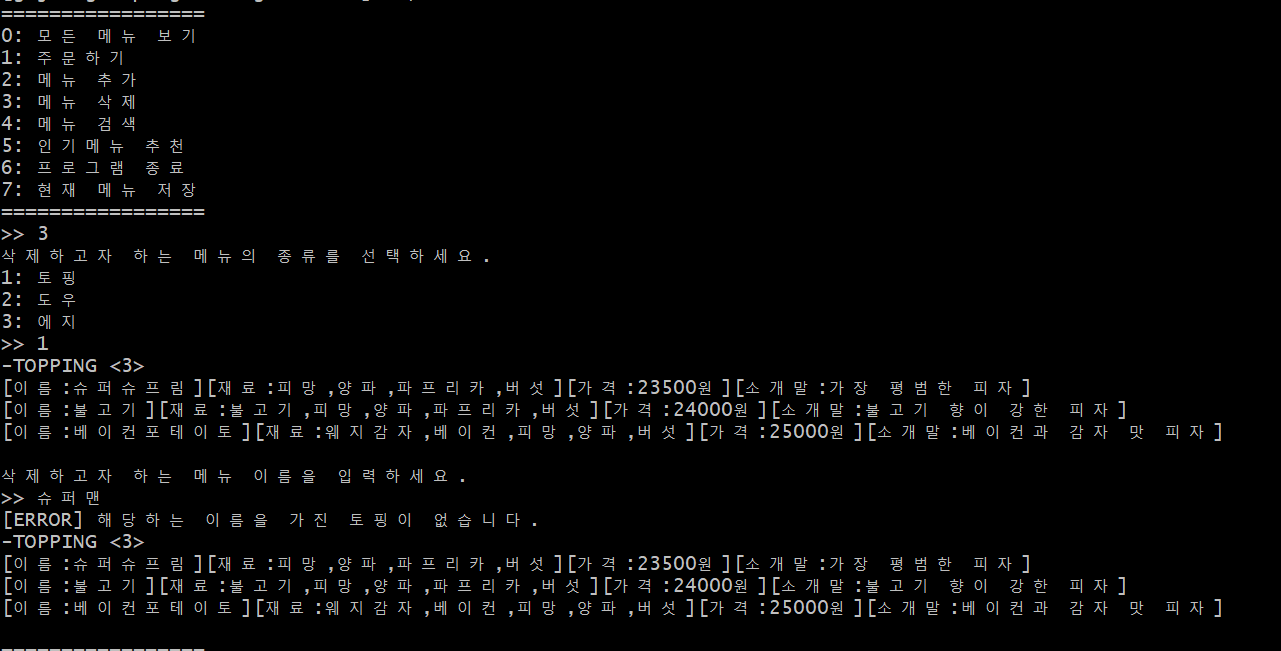
다음으로 add\_menu 함수는 메뉴 추가에 해당하는 함수로 ToppingList\*, DoughList\*, EdgeList\* 를 파라미터로 받는다. 사용자의 입력에 따라 토핑/도우/에지 중 하나를 추가하도록 한다. 예를 들어, 토핑의 경우 이름, 재료, 가격, 소개말을 차례로 입력받아서 이를 토핑 리스트에 추가한다. 이 때, 2개 이상의 재료의 경우 ‘,’ 를 통해 구분할 수 있도록 한다. 만약 가격 혹은 추가비용 입력이 정수가 아닐 경우, 도우의 두께가 float 형식이 아닐 경우 아래와 같이 에러 메시지를 출력하고 프로그램을 다시 실행한다.



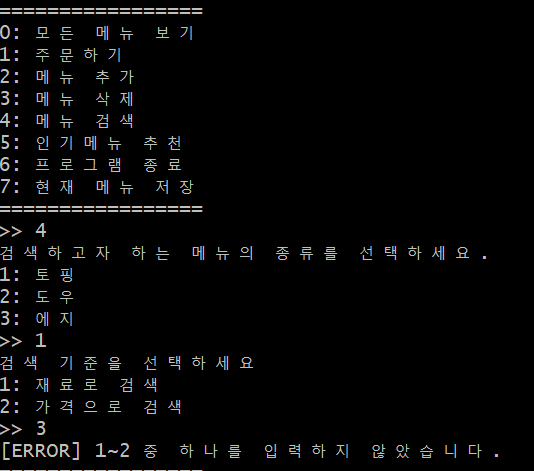
다음 설명할 함수는 del\_menu 함수다. del\_menu 함수 역시 파라미터로 ToppingList\*, DoughList\*, EdgeList\* 를 받는다. 사용자로부터 토핑, 도우, 에지 중 어떤 것을 삭제할 지 입력받는다. 이에 해당하는 1~3 중 하나를 입력하지 않으면 에러 메시지를 출력하고 함수 실행을 종료하고 다시 처음부터 사용자 명령을 받는다.



1~3을 입력받은 후, 원하는 이름을 입력하면 그 이름에 맞는 토핑/도우/에지를 리스트에서 삭제한다. 만약 해당하는 이름을 가진 토핑이 없으면 아래와 같이 에러 메시지를 출력하고 다시 처음부터 사용자 명령을 받는다.

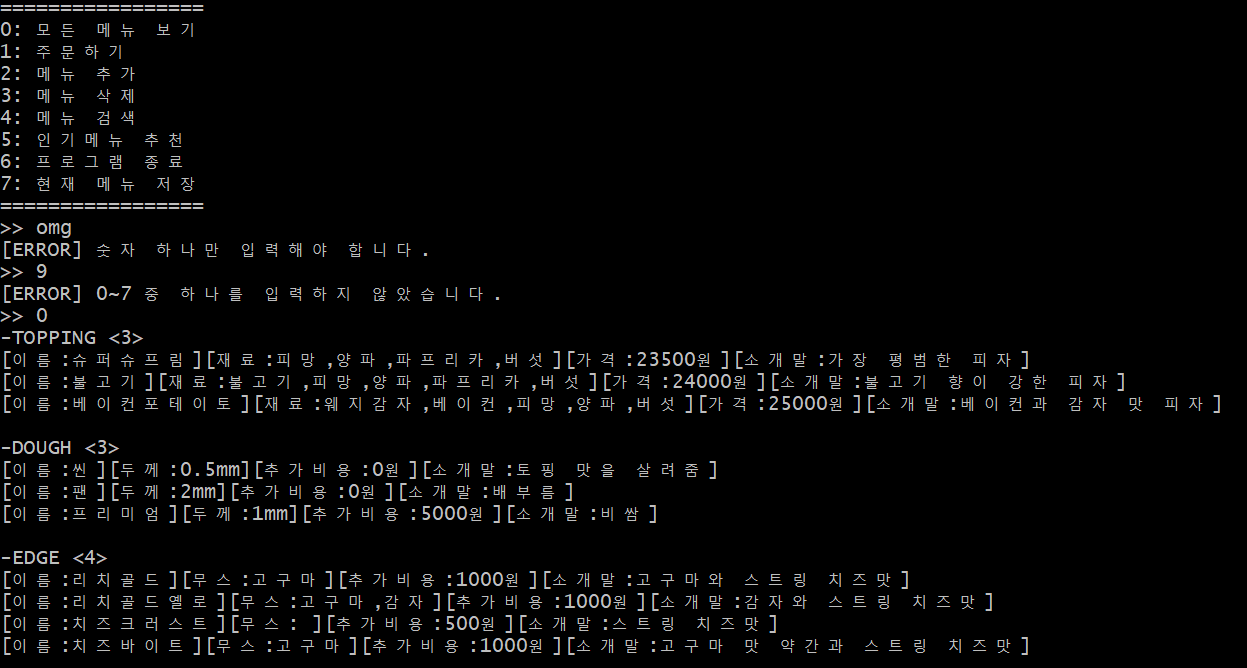


search\_menu 함수 또한 역시 파라미터로 ToppingList\*, DoughList\*, EdgeList\* 를 받는다. 먼저 토핑, 도우, 에지 중 하나를 선택하도록 1~3 중 하나를 입력받는다. Del\_menu 함수와 마찬가지로 1~3 중 하나를 입력하지 않으면 에러 메시지를 출력하고 함수 실행을 종료하고 다시 처음부터 사용자 명령을 받는다. 토핑은 재료로 검색과 가격으로 검색, 도우는 두께로 검색, 에지는 무스로 검색을 제공한다. 검색 기준의 범위 내 숫자를 입력하지 않으면 역시 에러 메시지를 출력하고 처음으로 돌아간다. 아래는 이와 관련된 예시이다.



read\_int 함수는 본 프로그램에서 숫자 하나를 읽는 경우가 많아서 만든 함수로, 사용자로부터 0~9 중 하나의 수를 입력 받아 이를 반환하는 함수다. 0~9 중 하나를 입력하지 않으면 에러 메시지를 출력하고 -1을 리턴한다. 이를 위해서 getline 함수를 이용해서 입력을 받고, 길이가 1이 아니거나 입력 값이 0이상 9이하가 아니면 에러 메시지를 출력한다.

print\_options 함수는 사용자의 명령 종류를 알려주고 0~6 중 하나를 입력받는다. 이를 입력받을 때, 위에서 설명한 read\_int 함수를 이용하며, read\_int 함수의 반환값이 -1이거나 0~6 중 하나가 아니면 문제가 있는 것이기 때문에 적절한 에러 메시지를 출력하고 0~6 중 하나를 입력할 때까지 다시 입력받도록 했다. 아래는 예시 화면이다.



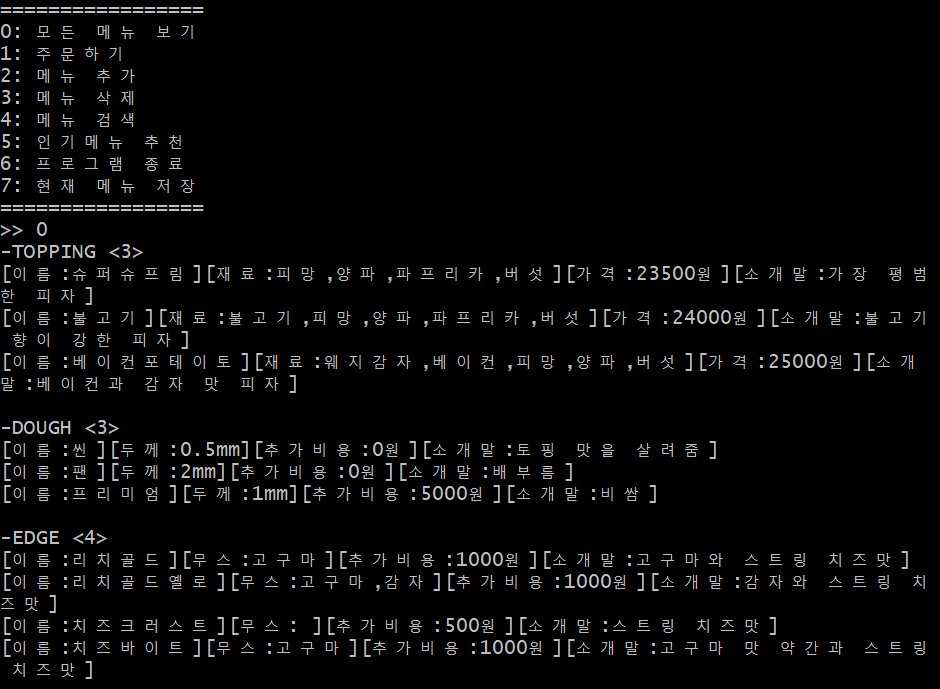
***마지막으로 save\_file 함수는 새롭게 만든 명령을 구현하기 위해서 사용하는 함수다.*** 이는 파라미터로 ToppingList\*, DoughList\*, EdgeList\* 를 받는다. save\_menu.txt 파일을 열고 –토핑 태그를 넣고 줄을 바꾼 후 파일을 닫는다. 파라미터로 받은 토핑 리스트의 save\_topping 함수를 실행하여 save\_menu.txt에 토핑 정보를 저장한다. 다음으로 다시 save\_menu.txt 파일을 열어서 –도우 태그를 넣고 줄을 바꾼 후 파일을 닫는다. 파라미터로 받은 도우 리스트의 save\_dough 함수를 실행하여 save\_menu에 도우 정보를 저장한다. 마지막으로 save\_menu.txt 파일을 열고, -에지 태그를 넣고 줄을 바꾼 후 파일을 닫는다. 파라미터로 받은 에지 리스트의 save\_edge() 함수를 이용하여 에지 정보를 저장하면 모든 정보의 저장이 끝난다. 이렇게 저장한 save\_menu.txt는 나중에 원할 때 init\_menu.txt로 이름만 바꾸면 처음에 빌드할 때 사용할 수 있다.

1. **프로그램 실행 방법**

해당 프로그램이 있는 폴더에서 make 커맨드를 통해 컴파일하면 main 파일이 생성된다. 이를 실행하면 사용자가 선택할 수 있는 옵션이 출력되고 사용자로부터 0~6 사이 입력을 받는다. 0은 모든 메뉴를 보여주고 1은 원하는 메뉴를 주문한다. 2는 원하는 메뉴를 추가하고 3은 원하는 메뉴를 삭제한다. 4는 특정 조건에 맞는 메뉴를 검색하고 5는 가장 많이 주문된 토핑/도우/에지를 각각 출력한다. 6을 입력하면 단순히 프로그램이 종료된다.

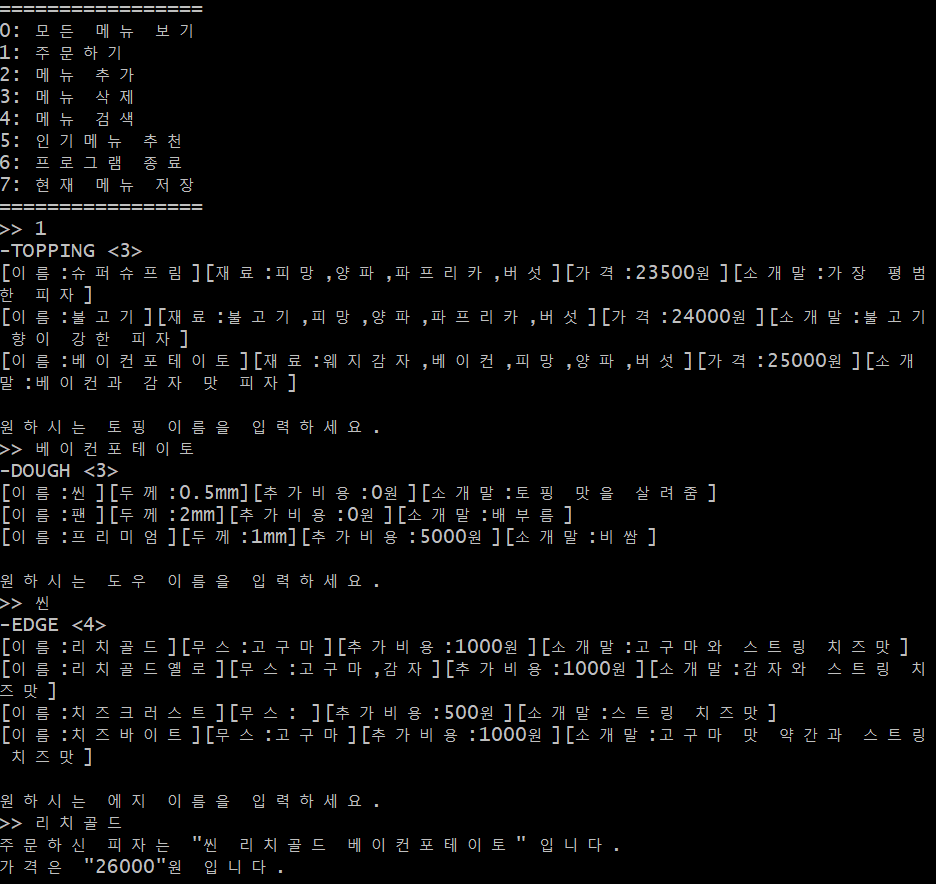
본 프로그램은 init\_menu.txt를 data file로 읽는다. 해당 이름의 파일이 없을 경우 에러 메시지를 출력하고 기존에 저장된 데이터 없이 프로그램을 실행한다. 해당 data file을 읽을 때는 위에서 설명한 read\_pizzas 함수를 구현하여 이용하였다. init\_menu.txt 의 형식에 따라 먼저 –토핑을 파싱하고, 탭을 기준으로 하여 토핑 이름/재료/가격/설명을 파싱한다. 재료는 따로 ‘,’를 기준으로 다시 파싱하여 하나 하나 토핑의 인스턴스에 넣어준다. 하나의 토핑 입력이 끝나면 해당 토핑의 주소를 토핑 리스트에 넣는다. ‘-도우’ 에 도달할 때까지 반복하여 모든 토핑이 입력되도록 한다. 도우 역시 토핑과 유사하게 진행되며, 특별히 다른 점은 도우에서는 float 형식의 두께를 받는다. ‘-에지’에 도달할 때까지 진행하며, 에지 입력 역시 도우/토핑과 유사하게 진행된다. 에지의 경우 무스를 여러 개 입력받을 수 있어 토핑의 재료와 같은 방식으로 ‘,’ 를 기준으로 파싱한다. 재료가 없는 경우는 ‘ ‘을 입력받아 이를 저장해둔다.

1. **프로그램 실행 예시**
2. **모든 메뉴 보기**



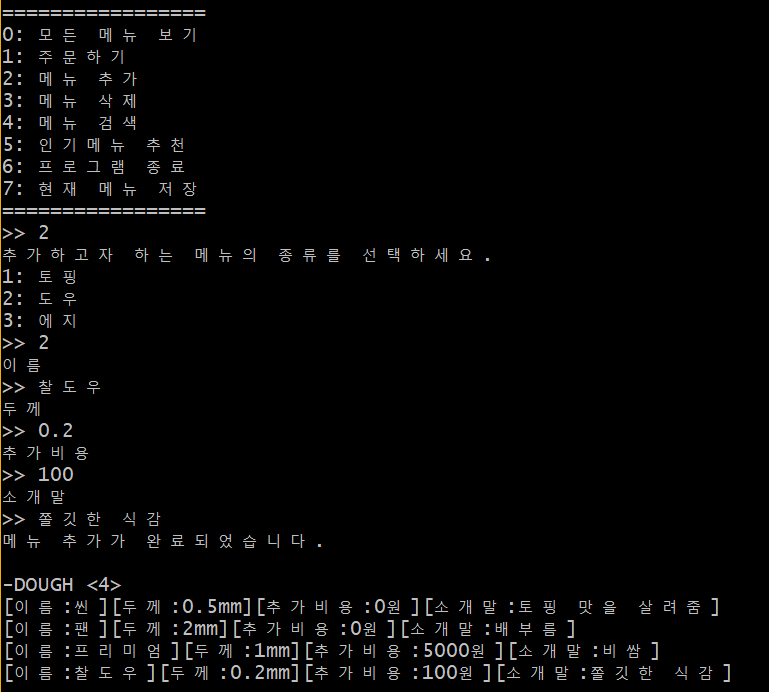
프로그램을 실행하고 0을 선택하면 위와 같은 화면이 나타난다.

1. **주문하기**



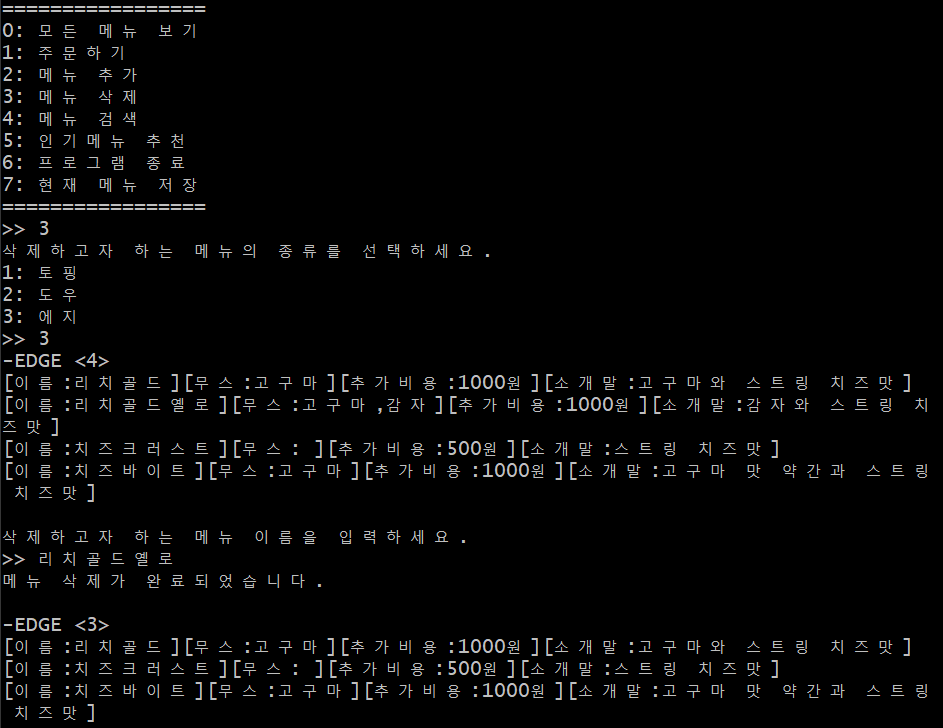
프로그램을 실행하고 1, 베이컨포테이토, 씬, 리치골드를 차례로 입력한 화면이다.

1. **메뉴 추가**



위 화면은 프로그램을 실행하고 2, 2, 찰도우, 0.2, 100, 쫄깃한 식감 을 차례로 입력하여 메뉴 추가를 한 화면이다.

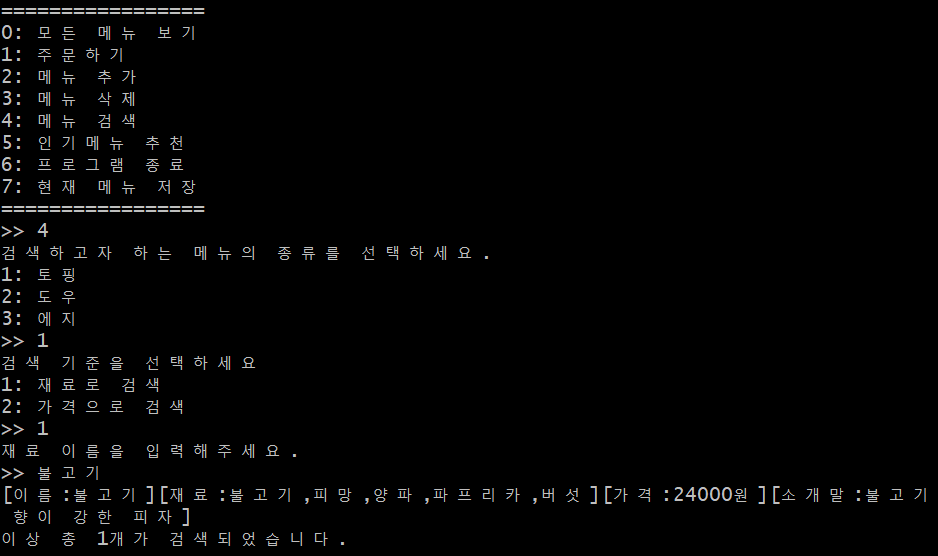
1. **메뉴 삭제**



위 화면은 프로그램을 실행하고 3, 3, 리치골드옐로 를 차례로 입력하여 해당하는 에지 메뉴를 삭제한 화면이다.

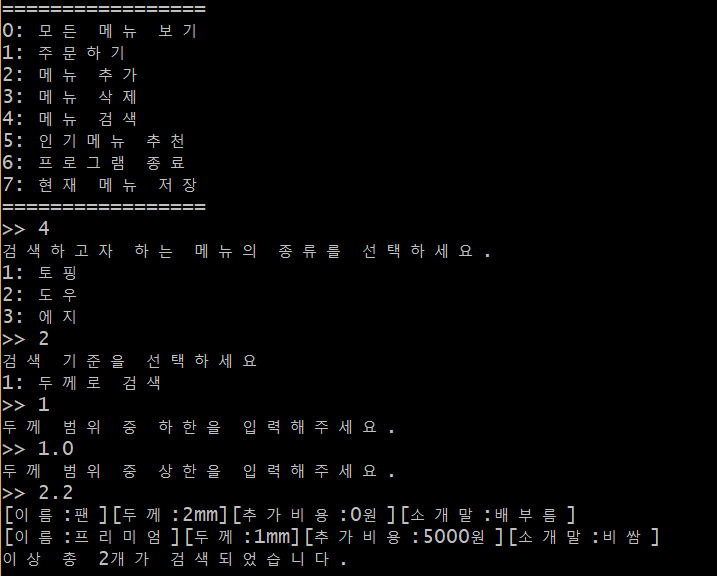
1. **메뉴 검색**

**<4-1> 재료로 메뉴 검색**



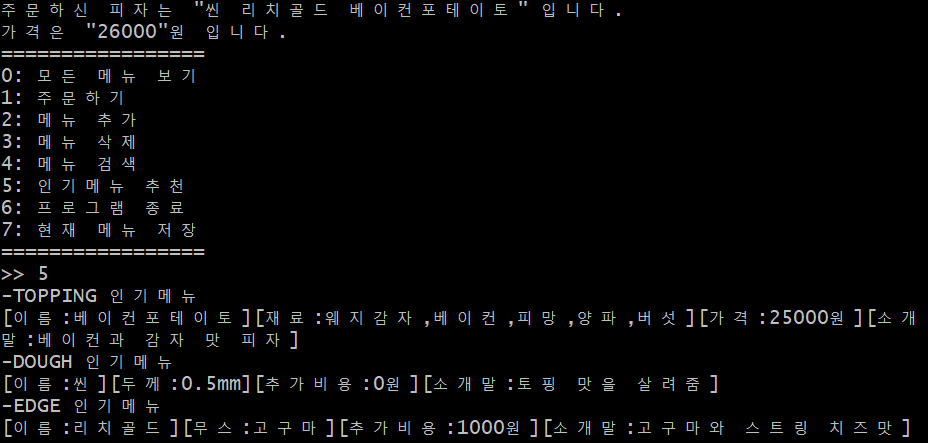
위 화면은 4, 1, 1, 불고기를 입력하여 불고기를 재료로 포함하는 토핑들을 모두 출력하는 화면이다.

**<4-2> 두께로 메뉴 검색**



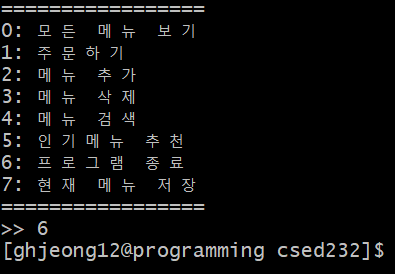
위 화면은 4, 2, 1, 1.0, 2.2 를 차례로 입력하여 두께가 1.0에서 2.2 사이인 도우들을 출력하는 화면이다.

1. **인기 메뉴 추천**



위 화면은 씬 리치골드 베이컨포테이토를 한번 주문한 다음 5를 입력하여 인기메뉴 추천을 실행한 화면이다.

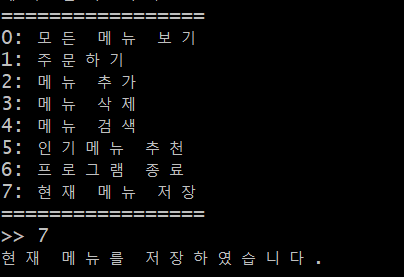
1. **프로그램 종료**



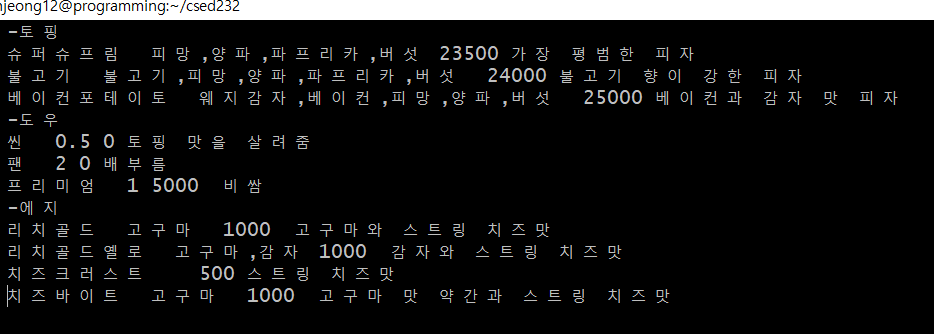
위 화면은 6을 입력하여 프로그램을 종료한 화면이다.

1. **현재 메뉴 저장**

***본 기능은 과제 문서에 없는 7번 명령에 해당하는 새로 추가한 기능이다.***



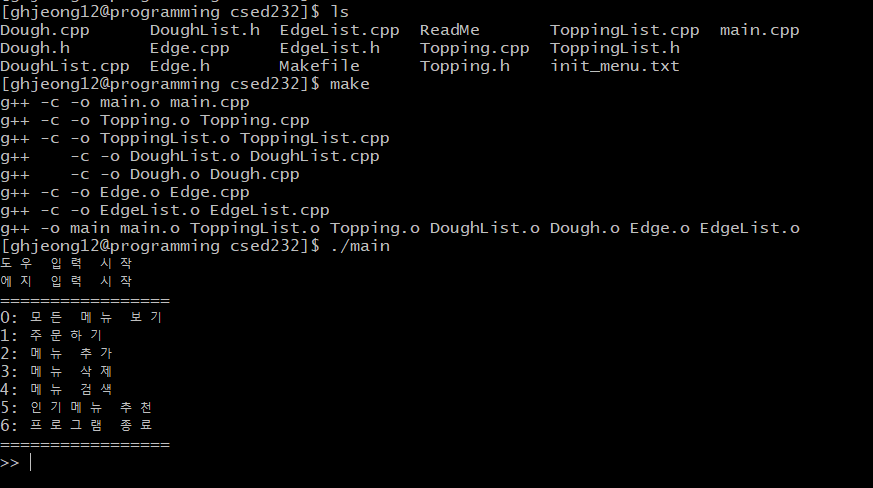
아래는 save\_menu.txt에 저장된 결과다. 이를 init\_menu.txt 로 재설정하고 실행하면 기존의 init\_menu.txt와 같은 결과가 나온다. 본 함수를 이용하면, 메뉴 추가나 삭제를 한 시점에서 이에 해당하는 토핑/도우/에지 정보들을 save\_menu.txt 파일로 저장할 수 있다. 나중에 실행할 때 save\_menu.txt를 init\_menu.txt로 바꿔서 실행하면 저장한 정보들을 불러올 수 있다.



1. **위에서 설명되지 않은 요구사항**

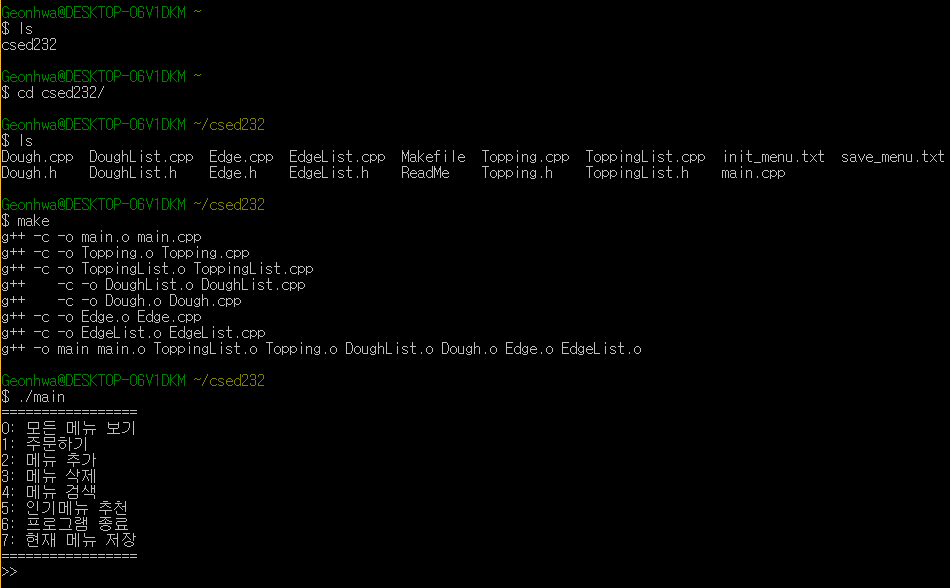
**Linux 및 Windows 에서의 프로그램 컴파일 및 실행과정**

먼저 리눅스에서 컴파일 및 실행과정은 아래와 같다.



프로그래밍 실습 서버에서 .cpp 와 .h 가 있는 디렉토리에서 make 를 실행하면 Makefile 에 따라 컴파일이 진행되고, main 실행파일이 생성된다. 그 후 ./main 커맨드를 이용하여 main 파일을 실행하면 프로그램이 실행되는 것을 확인할 수 있다.

다음으로 Windows에서 mingw32를 이용해 실행해보았다. 먼저 프로그래밍 실습 서버에 있는 csed232 폴더를 다운받아 mingw/msys/1.0/home/Geonhwa 에 옮긴다. 그런 다음 해당 위치에서 ls 를 이용하면 csed232가 생긴 것을 확인할 수 있다. 역시 make를 이용하여 컴파일하고 ./main 커맨드를 이용해 실행 가능하다. 아래 사진은 그 과정을 보여준다.



1. **토론, 결론 및 개선 방향**

이번 과제를 통해서 클래스를 이용하여 객체지향적으로 프로그래밍하는 연습을 하였다. 프로그래밍 실습 서버에서 작업하여 리눅스 환경에서 c++ 을 이용하고, makefile 을 통해 컴파일하는 연습 또한 할 수 있었다. 프로그램을 짜면서 최대한 가독성을 높이고 유지 보수가 쉽도록 하고자 생각하면서 진행했다. main 함수 내부를 최대한 직관적으로 하고, 코드가 길어지는 부분을 따로 함수로 추출해내어 간단히 main 함수만 보아도 전체적으로 어떻게 프로그램이 진행되는지 알 수 있도록 하였다.

과제 설명이 잘 되어 있어 특별히 문제가 생기는 경우는 별로 없었다. 한 가지, 개발 과정에서 입력 버퍼에 남아있는 값 때문에 입력 값을 받을 때 원하지 않는 값이 들어오는 경우가 종종 있었다. \n 이 남아서 다시 들어가는 것으로 보였으며, 이를 해결하기 위해 cin.ignore() 을 이용하였다.

**추가적으로 리눅스 서버에서는 init\_menu.txt에서 잘 입력을 받는데 Windows에서 실행했을 때는 문제가 생기는 경우가 있었다.** 확인해본 결과 .txt 파일을 생성하여 저장할 때 줄바꿈을 리눅스와 Windows에서 다르게 인식하여 생긴 문제였다. 주어진 init\_menu.txt를 리눅스에서 줄 단위로 읽으면 마지막에 아스키코드 13번이 추가된다. 윈도우에서는 뉴라인이 \r\n 인데 리눅스에서는 \n 으로 표시되기 때문인 것으로 파악된다. 이를 처리하기 위해 만약 한 줄을 읽은 시점에서 마지막 문자가 아스키코드 13번이면 substr 함수를 이용해서 마지막 한 문자를 버렸다.

이 프로그램은 개선의 여지가 많다. 프로그램의 요구사항 중 하나가 상속을 사용하지 않는 것이었다. 이에 따라 상속을 이용하지 않았지만, 만약 상속을 이용한다면 토핑, 도우, 에지에서 겹치는 부분들을 하나의 클래스로 묶어낼 수 있어 중복되는 코드를 줄일 수 있을 것이다. 또한 링크드리스트의 경우도 링크드리스트의 요소(element)의 타입만 다를 뿐 실제로 같은 기능을 한다. 따라서 각 타입에 따라 3개의 각기 다른 리스트가 아니라 하나의 링크드리스트 클래스를 이용하면 중복되는 코드의 양을 줄일 수 있고, 유지 보수 측면에서도 좋을 것이다.