객체지향프로그래밍 -1-2주차 실습 활동지 (2019년 9월 3일/10일)

성명:안창희 학과: 소프트웨어 학번:201723272 반/그룹: 318호 반

※ 본 실습활동지를 작성함에 있어 다른 학생의 문서로부터 일부 또는 전체를 복사하였습니까
예( ) 아니오(O) (복사 하였다면 예에 체크하고 아니라면 아니오에 체크하시오)

1. 다음 지문을 읽고 물음에 답하시오.
간단한 인사관리 프로그램을 작성하고자 한다. 이 프로그램은 다음과 같은 기능을 수행할 수 있다. 사원의 정보는 사번(고유한 번호), 성명, 급여로 이루어져 있다고 가정한다. 사번은 다섯 자리 정수형(11111-99999), 성명은 문자열, 급여는 실수라고 가정한다.
1. 사원 목록 보기
- 모든 사원의 정보를 보여준다.
2. 사원 추가하기
- 주어진 사원의 정보를 이용하여 한명의 사원을 사원목록에 추가한다.
3. 한 사원의 급여 인상하기
- 주어진 사원(id)의 급여를 주어진 퍼센트만큼 인상한다.
4. 전체 사원 급여 인상하기
- 모든 사원의 급여를 주어진 퍼센트만큼 인상한다.
프로그램을 시작하면 위 기능을 선택할 수 있는 메뉴를 보여준다. 하나의 기능의 수행이 종료하면 다시 메뉴를 보여준다.

1) 위 프로그램에서 어떤 데이터가 존재하는 지 파악하고, 각 데이터를 어떤 데이터 구조로 표현하면 좋을지 설명하시오. (이 과제에서는 object-oriented programming 기법을 사용하지 않는다)

위 프로그램을 C언어로 구현하기 위해서 내가 필요하다고 생각하는 데이터는 다음과 같다

-정수형데이터 “사번”
-정수형데이터 “급여”
-문자배열데이터 “이름”
-사번, 급여, 이름을 포함하는 구조체 “사원”(여러 명의 사원을 저장하기 위해 배열로 구현)
-관리할 수 있는 최대 사원수이자 정수형데이터 “MAX사원”
-현재까지 관리중인 사원수이자 정수형데이터 “Current사원”
-MAX사원, Current사원, 사원 구조체 배열을 포함하는 “사원리스트”

프로그램의 계획 단계에서는 배열리스트 구조와 연결리스트 구조의 특성을 두고 고민했으며,
코드 구현의 간단화와 각 사원 데이터의 접근을 수행하는 속도의 최적화를 위해서 배열리스트 구조의 방식을 차용했다.

2) 위 문제를 가능한 한 많은 독립적인 function들로 분할하여 작성하려고 한다. 어떤 function이 존재하는지 말하고(이름과 기능) 각 function이 위 문제 1번에서 밝힌 data 중 어떤 data를 이용하는 지 말하시오(표나 그림으로 정리).

1. StaffList* createStaffList(int MAX_Staff); 사원리스트 생성함수

“사원 리스트“ 생성함수 createStaffList는 1)에서 명시한 “MAX사원”을 받아 그 수만큼의 사원수를 가진 “사원 리스트“를 생성하는 함수이다. 이후 사원 리스트를 가리키는 포인터 주소를 반환한다.

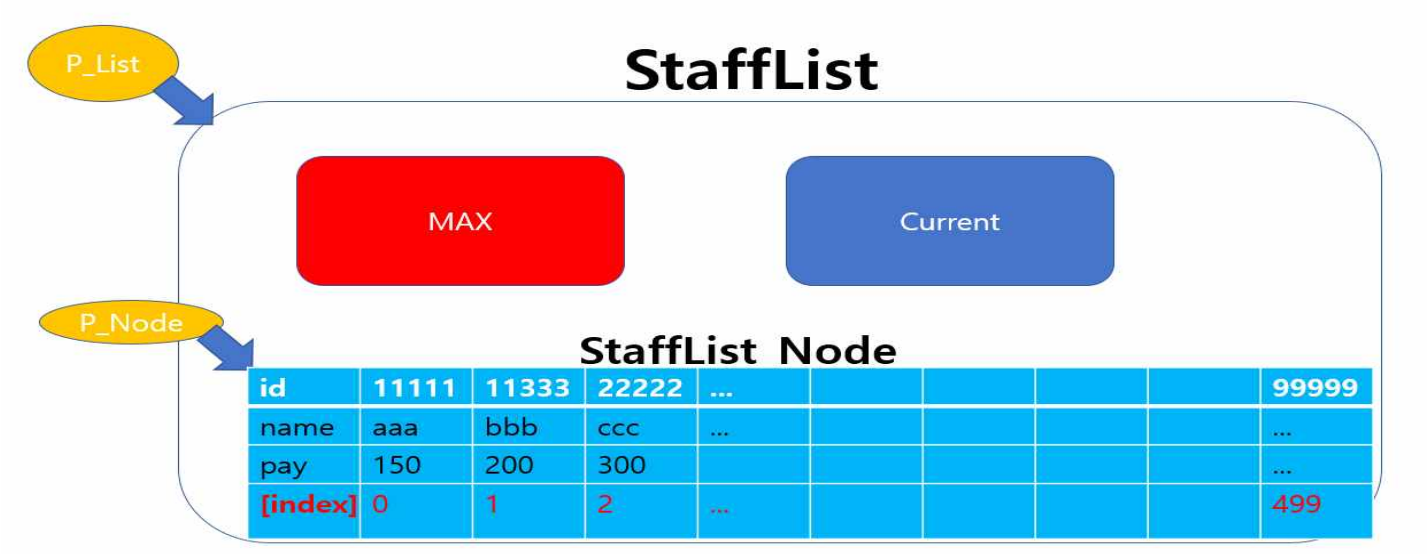
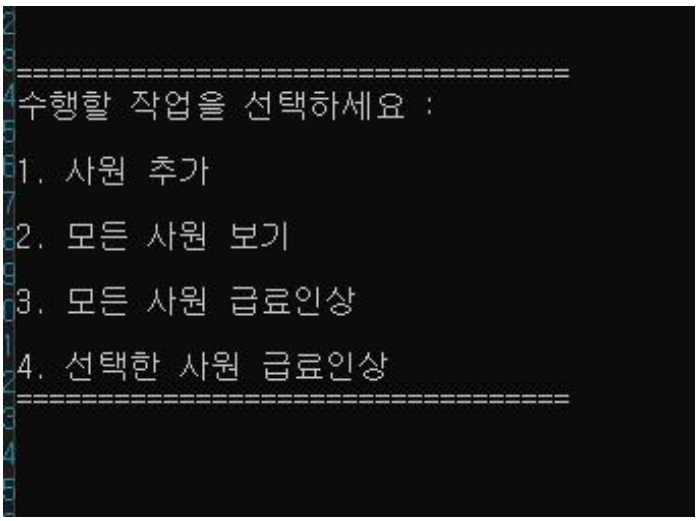


그림1) 추상데이터 “사원 리스트”와 “사원”의 구조를 나타낸 자료.
@“MAX사원“의 값은 500으로 가정하였다.

그림2) 초기실행화면



2.void addStaff(StaffList* P_List, int position, StaffList_Node newstaff); 사원 추가함수

사원 추가함수 addStaff는 “사원 리스트”의 포인터 값인 P_List, “사원“배열에서 입력을 받을 위치인 position, 그리고 새로 생성할 ”사원“정보를 임시로 저장해둘 “사원”구조체인 newstaff, 이렇게 세 개의 값을 입력받는다.
이후 position에 해당되는 인덱스의 “사원”구조체에 newstaff에 해당하는 “사원”정보를 그대로 복사한다.
(newstaff의 “사번”, “이름”, “급여”에 해당하는 값은 메인 함수 내에서 미리 입력받은 상태이며 addStaff의 인자로서 위의 설명과 같이 가동한다.) 함수 종료 시 입력완료 메시지를 출력한다.

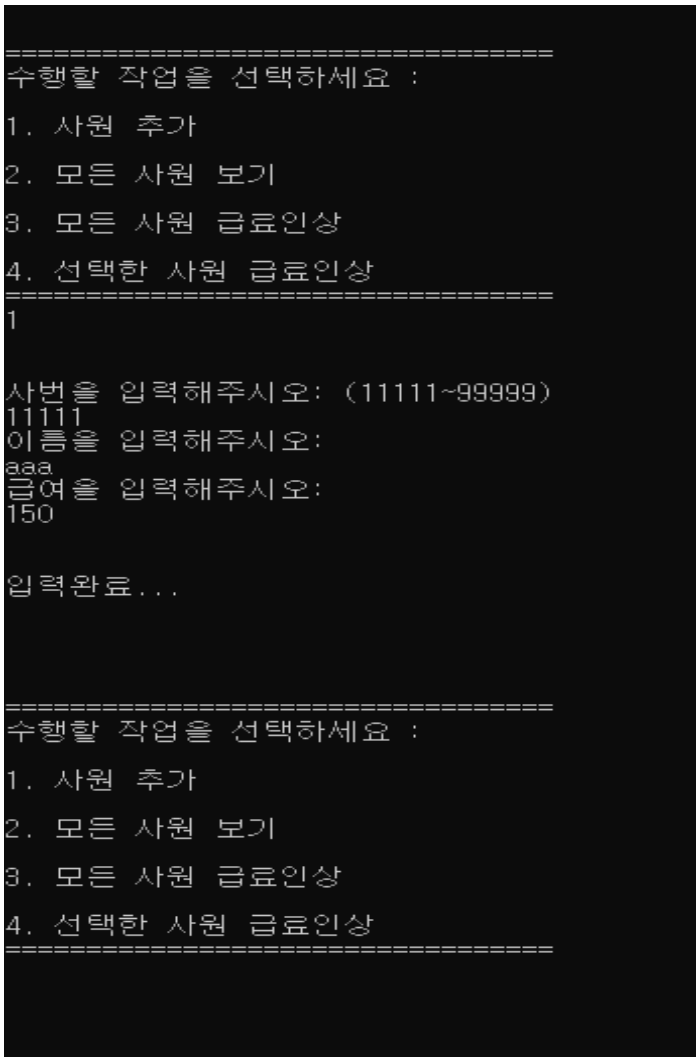


그림3) 사원추가 실행화면

3.StaffList_Node* getStaff(StaffList* P_List, int position); 사원정보 접근함수

사원정보 접근함수인 getStaff는 “사원 리스트”의 포인터 값인 P_List, 접근할 위치인 position을 입력받아서
접근할 위치의 인덱스에 해당하는 “사원” 포인터주소인 StaffList_Node* 형 값을 반환하는 함수이다.
이 함수는 직접적으로 사용되지 않고, 다른 함수에서 특정 인덱스의 “사원”정보에 접근하고 싶을 때 활용하는 함수이다.

4.void showall(StaffList* P_List, StaffList_Node* P_Value); 사원리스트 전체일람 함수

사원리스트 전체일람 함수인 showall은 P_List, P_Value값 두 개를 받아서 전체 “사원“정보를 각 사원별로
“사번”, “이름”, “급여” 순으로 한 줄씩 모두 출력한 후, "MAX사원"대비 “Current사원”값을 출력하는 함수이다.
함수 종료 시 출력완료 메시지를 출력한다.

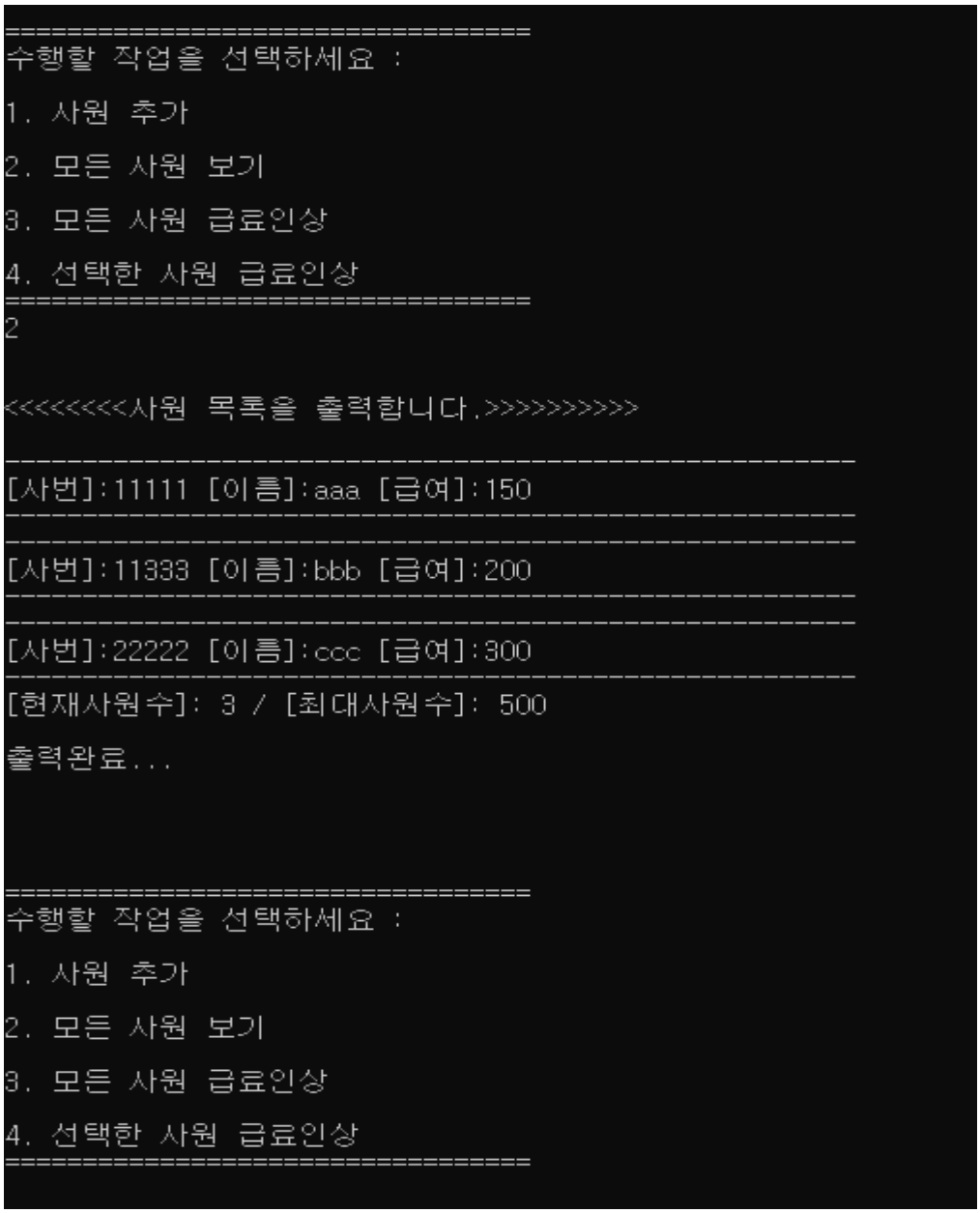


그림4) 그림1에서 나온 자료대로 사원을 추가 후 “사원리스트”를 전체일람 했을 때의 실행화면

5.void payup_all(StaffList* P_List, StaffList_Node* P_Value);
전체사원 급여인상 함수인 payup_all은 P_List와 P_Value값을 입력받아서 for문을 통해 0부터 “Current사원”까지에 해당하는 인덱스의 전체 사원들의 각각 급여에 입력된 퍼센트만큼의 급여를 인상시키는 연산을 실행한 후 종료된다.
함수 종료 시 전체인상완료 메시지를 출력한다.

그림5) 그림4 이후 전체사원의 급여를 인상한 화면

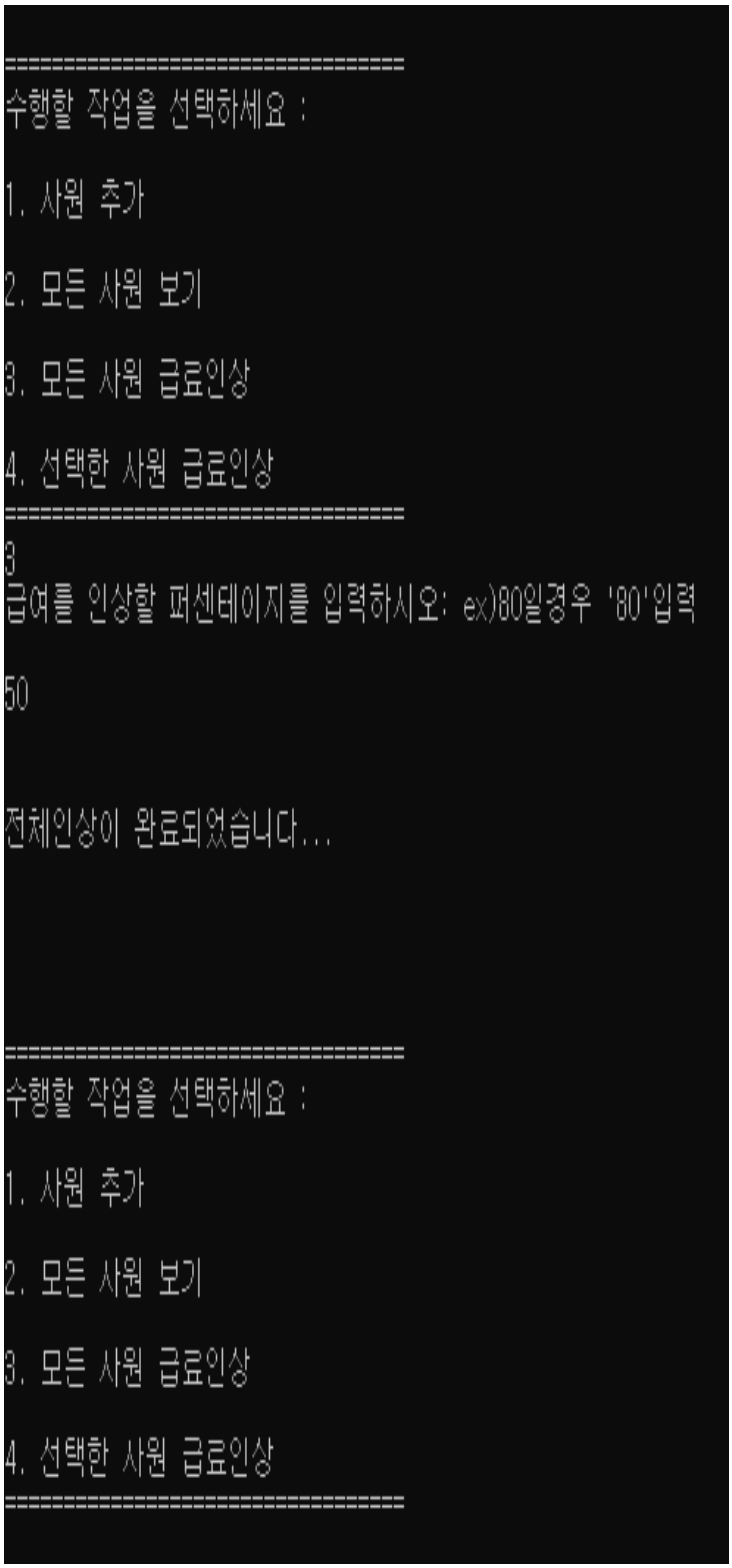
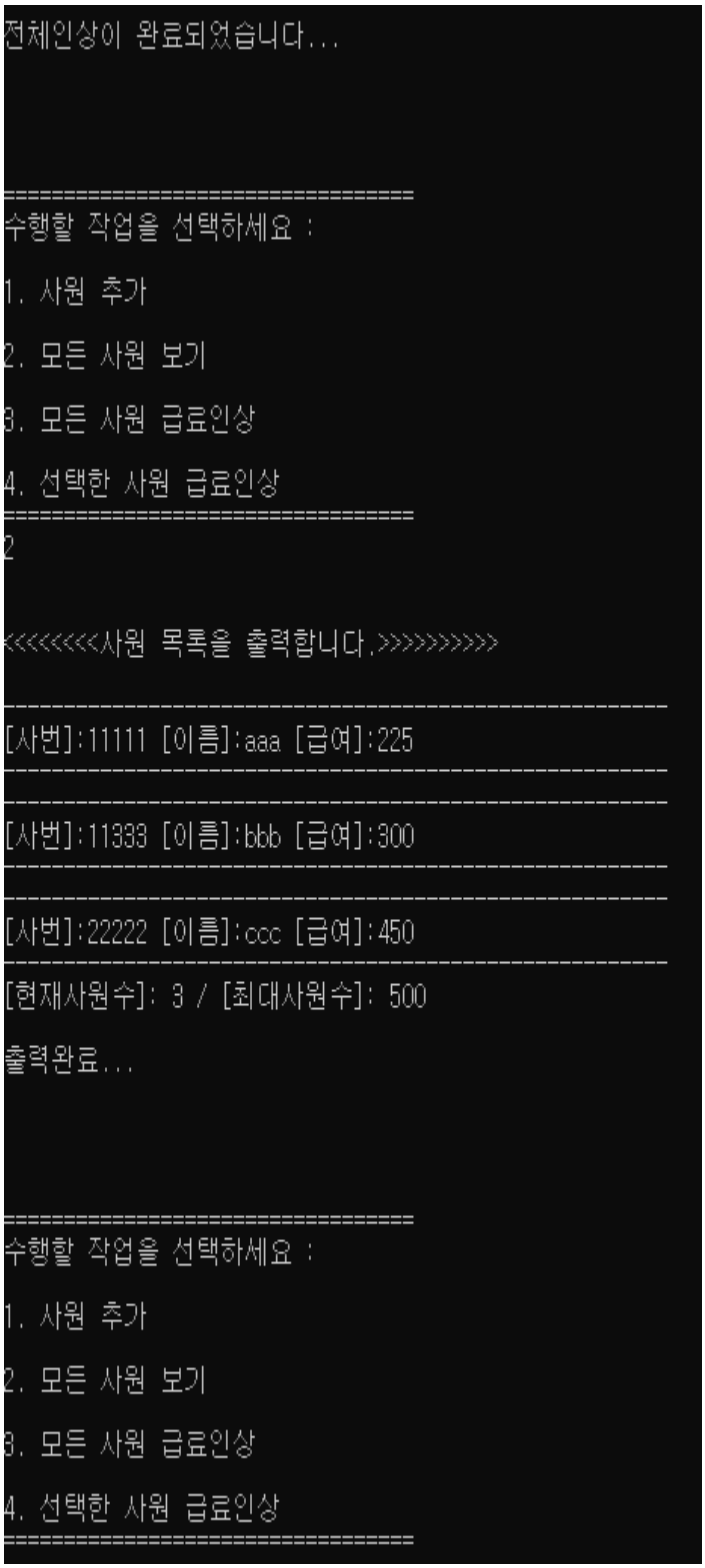


그림6) 그림5의 전체 인상을 마치고 출력한 “사원리스트”



6.void payup_one(StaffList* P_List, StaffList_Node* P_Value);
단일사원 급여인상 함수인 payup_one은 P_List와 P_Value값을 입력받아서 for문을 통해 0부터 “Current사원”까지에 해당하는 인덱스의 사원들의 전체 “사번“과 함수실행 중 입력받은 급여인상 대상 “사원“의 "사번"이 일치하는지 모두 비교하고, 일치하는 “사번”에 해당하는 “사원”의 “급여”에만 입력된 퍼센트만큼의 급여를 인상시키는 연산을 실행한 후 종료된다.
함수 종료 시 선택인상완료 메시지를 출력한다.

그림7) 그림6의 전체인상 실행 후,
a "사원"에 대해서만 100퍼센트 급여인상을 한 번 더 실행하는 화면(a사원의 선택은 사번인 11111을 사용)

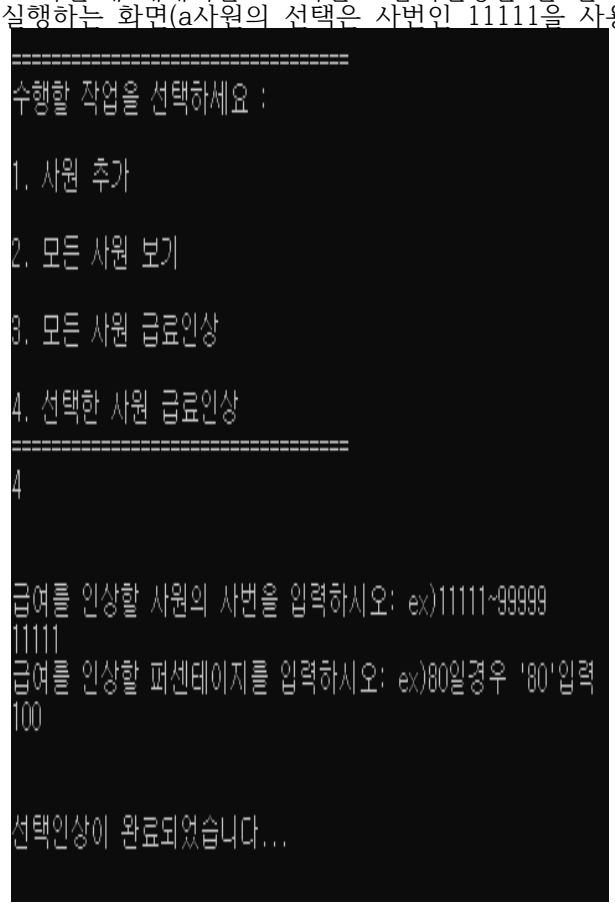
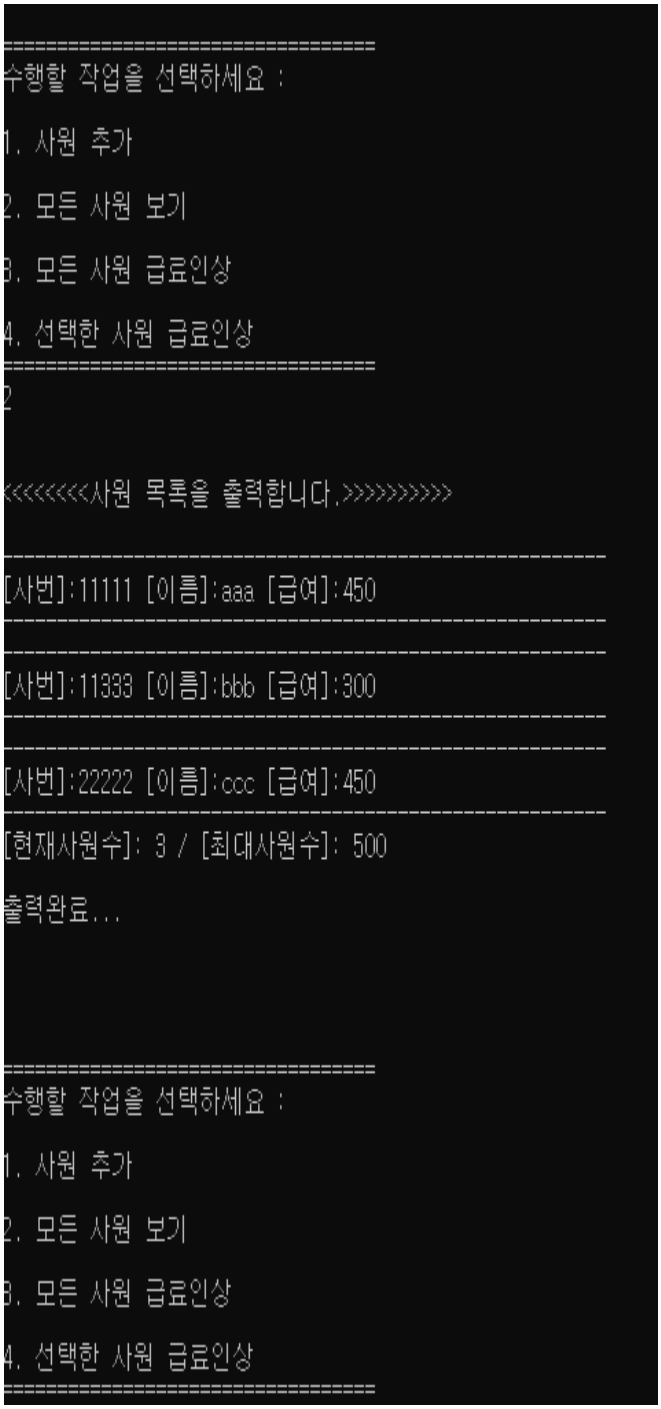


그림8) 그림7의 선택인상 실행 후 출력한 “사원 리스트”



[프로그램 전체코드]

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

typedef struct StaffList_Node {
int id;
char name[20];
int pay;
}StaffList_Node

typedef struct StaffList {
int MAX_Staff;
int Current_Staff;
StaffList_Node* P_Node;
}StaffList

StaffList* createStaffList(int MAX_Staff);
void addStaff(StaffList* P_List, int position, StaffList_Node newstaff);
StaffList_Node* getStaff(StaffList* P_List, int position);
void showall(StaffList* P_List, StaffList_Node* P_Value);
void payup_all(StaffList* P_List, StaffList_Node* P_Value);
void payup_one(StaffList* P_List, StaffList_Node* P_Value);

int main()
{
StaffList* P_List = NULL
StaffList_Node* P_Value = NULL
int select;
P_List = createStaffList(500); // 최대인원500의리스트생성
while (1)
{
printf("\n\n\n================================== \n");
printf("수행할작업을선택하세요: \n\n");
printf("1. 사원추가\n\n");
printf("2. 모든사원보기\n\n");
printf("3. 모든사원급료인상\n\n");
printf("4. 선택한사원급료인상\n");
printf("================================== \n");
scanf_s("%d", &select);


switch (select)
```

```
{
case 1://사원추가
if(P_List != NULL)
{
StaffList_Node temporary;
printf("\n\n사번을입력해주시오: \n");
scanf_s("%d", &temporary.id);

//사번의범위제한
if (temporary.id < 11111 || temporary.id >99999)
{
fprintf(stderr, "오류: 사번은11111부터99999사이의수로입력되어야합니다");
exit(1);
}



//사번의중복검사
for (int i = 0; i < P_List->Current_Staff; i++)
{
P_Value = getStaff(P_List, i);
if (temporary.id == P_Value->id )
{
fprintf(stderr, "오류: 중복되는사번이존재합니다");
exit(1);
}
}

printf("이름을입력해주시오: \n");
scanf_s("%s", temporary.name, 20);
printf("급여을입력해주시오: \n");
scanf_s("%d", &temporary.pay);

addStaff(P_List, P_List->Current_Staff, temporary);
}
continue

case 2: //사원전체목록출력
showall(P_List, P_Value);
continue

case 3://사원전체percent만큼월급인상
payup_all(P_List, P_Value);
continue

case 4://사원한명percent만큼월급인상
```

```
payup_one(P_List, P_Value);
continue
}
}
return 0;
}




StaffList* createStaffList(int MAX_Staff)
{
StaffList* pReturn = NULL


if (MAX_Staff > 0)
{
pReturn = (StaffList*)malloc(sizeof(StaffList));

if (pReturn != NULL)
{
pReturn->MAX_Staff = MAX_Staff
pReturn->Current_Staff = 0;
pReturn->P_Node = NULL
}
else
{
printf("오류, 메모리할당createStaffList()\n");
return NULL
}
}
else
{
printf("오류, 최대사원수는0이상이어야합니다\n");
return NULL
}

pReturn->P_Node = (StaffList_Node*)malloc(sizeof(StaffList_Node) * MAX_Staff); //StaffList_Node 를MAX_Staff 개
수만큼만든다
if (pReturn->P_Node == NULL)
{
printf("오류, 2번째메모리할당createArrayList()\n");
free(pReturn); return NULL
}
memset(pReturn->P_Node, 0, sizeof(StaffList_Node) * MAX_Staff); //StaffList_Node 전체의데이터값을0으로초기화한
```

다.

```
return pReturn;
}

void addStaff(StaffList* P_List,int position, StaffList_Node newstaff)
{
if (P_List != NULL)
{
P_List->P_Node[position] = newstaff //지정한위치에삽입할구조체추가
}
P_List->Current_Staff++;

printf("\n\n");
printf("입력완료...");
printf("\n\n");
return
}

StaffList_Node* getStaff(StaffList* P_List, int position)
{
StaffList_Node* P_Return = NULL
if (P_List != NULL)
{
P_Return = &(P_List->P_Node[position]);
}
return P_Return;
}

void showall(StaffList* P_List, StaffList_Node* P_Value)
{
if (P_List != NULL)
{
printf("\n\n<<<<<<<<<사원목록을출력합니다.>>>>>>>>>>\n\n");
for (int i = 0; i < P_List->Current_Staff; i++)
{
P_Value = getStaff(P_List, i);
printf("-----------------------------------------------------\n");
printf("[사번]:%d [이름]:%s [급여]:%d \n", P_Value->id, P_Value->name, P_Value->pay);
printf("-----------------------------------------------------\n");
}
printf("\n\n");
printf("출력완료...");
printf("\n\n");
}
```

```
}

void payup_all(StaffList* P_List, StaffList_Node* P_Value)
{
if (P_List != NULL)
{
int percent;
printf("급여를인상할퍼센테이지를입력하시오: ex)80%일경우'80'입력\n\n");

scanf_s("%d", &percent);



for (int i = 0; i < P_List->Current_Staff; i++)
{
P_Value = getStaff(P_List, i);
P_Value->pay = P_Value->pay * ((float)percent / 100) + P_Value->pay;
}
}
printf("\n\n");
printf("전체인상이완료되었습니다...");
printf("\n\n");
}


void payup_one(StaffList* P_List, StaffList_Node* P_Value)
{
if (P_List != NULL) //사원하나percent만큼월급인상
{
int percent;
int id;

printf("\n\n급여를인상할사원의사번을입력하시오: ex)11111~99999\n");

scanf_s("%d", &id);

printf("급여를인상할퍼센테이지를입력하시오: ex)80%일경우'80'입력\n");

scanf_s("%d", &percent);

for (int i = 0; i < P_List->Current_Staff; i++)
{
P_Value = getStaff(P_List, i);
if (P_Value->id == id) //사번이일치할경우
{
```

```
P_Value->pay = P_Value->pay * ((float)percent / 100) + P_Value->pay;
}
}
}

printf("\n\n");
printf("선택인상이완료되었습니다...");
printf("\n\n");
}
```