학번 : _____ 성명 : _____.

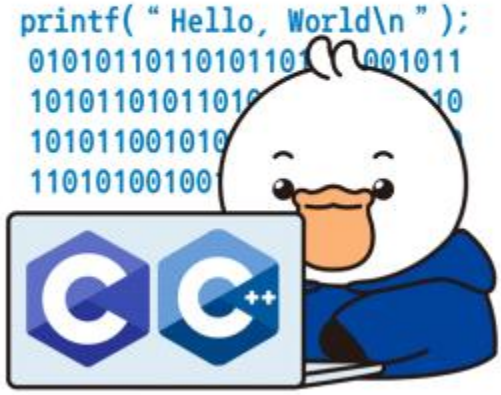
- 본 강의자료는 과학기술정보통신부 및 정보통신기획평가원에서 지원하는『소프트웨어중심대학』사업의 결과물입니다.
- 본 강의자료는 내용은 전재할 수 없으며, 인용할 때에는 반드시 과학기술정보통신부와 정보통신기획평가원의 '소프트웨어중심대학'의 결과물이라는 출처를 밝혀야 합니다.

SW 대구가톨릭대학교 소프트웨어중심대학사업단

대구가톨릭대학교 컴퓨터소프트웨어학부 School of Computer Software, Daegu Catholic University

Part 11.
고급 자료형과 전처리 활용

목차



11.1 구조체

- 구조체 [ ]
  - ✓ [ ]이 있는 다양한 데이터 타입의 변수들을 [ ]로 묶어 새로운 데이터 타입 형태로 정의 하는 것
  - ✓ 서로 [ ]이 높은 변수들을 하나로 통합하여 사용할 수 있도록 해주는 대표적인 [ ](derived) 데이터 타입
  - ✓ [ ]
    - ➢ 구조체에 묶여 있는 각각의 변수
    - ➢ 변수, 포인터, 배열, 공용체 등 모든 종류의 변수와 자료구조 사용 가능
  - ✓ 구조체 사용하기
    - ➢ [ ]에 구조체를 정의
    - ➢ 정의된 구조체를 이용하여 [ ]를 선언

11.1 구조체

- [ ] 및 [ ] 선언 과정

```
struct 구조체이름{
   멤버변수1 선언;
   멤버변수2 선언;
   …
   멤버변수n 선언;
}변수이름;
```

```
int main(){
  struct contact{
     char user_name[7];
     char phone_number[14];
     int global_number;
  };

  struct contact con_1;

  return 0;
}
void test(){
  struct contact con_1;
}
```

(a)

```
struct contact{
   char user_name[7];
   char phone_number[14];
   int global_number;
};

int main(){
  struct contact con_1;

  return 0;
}
void test(){
  struct contact con_1;
}
```

(b)

11.1 구조체

- 구조체 정의
  - ✓ [ ] 키워드 작성 후 구조체 [ ] 작성
  - ✓ 구조체 [ ]
    - ➢ [ ]가 정의하는 새로운 데이터 타입으로 사용 가능
    - ➢ [ ] 생성 규칙에 따라 정의되어야 함
    - ➢ [ ]라고 부름

11.1 구조체

- 구조체 정의
  - ✓ [ ] 변수
    - ➢ [ ] 작성 후, [ ] 안에 필요한 멤버 변수 [ ] 작성
    - ➢ 각 [ ]의 데이터 [ ]과 [ ] 작성하여 [ ] 변수 선언
    - ➢ 변수, 배열, 포인터 등 모든 형태 가능
      - 구조체 정의는 새로운 데이터 타입을 정의하는 단계로 실제 메모리에 존재x
      - 멤버 변수 [ ] 불가능
  - ✓ 구조체 정의 종료
    - ➢ 중괄호 이후 [ ] 작성
    - ➢ [ ]과 [ ] 사이에 [ ] 작성 - 구조체 정의와 동시에 변수 선언

11.1 구조체

- 정의한 구조체를 이용하여 [ ]
  - ✓ [ ] 구조체
    - ➢ 해당 지역, 즉 함수나 [ ] 내부에서만 정의가 유효
  - ✓ [ ] 구조체
    - ➢ 해당 코드 [ ]에 유효한 구조체
  - ✓ 구조체 변수 선언을 통해서 [ ]에 공간을 할당
    - ➢ 구조체 정의 후 변수 선언하지 않을 경우
      - 실제 메모리에 존재하지 않는 [ ]만 가짐

11.1 구조체

소스 11-1

```c
#include <stdio.h>
#include <string.h>
struct contact {
    char userName[20];
    char phoneNumber[15];
    int globalNumber;
}con1;
int main(void) {
    struct contact con2 = {"Gildong Hong", "010-1234-5678", 82};
    strcpy(con1.userName, "Steve Jobs");
    strcpy(con1.phoneNumber, "000-000-0000");
    con1.globalNumber = 1;
    printf("Size of struct contact : %lu\n", sizeof(struct contact));
    printf("Size of userName : %lu\n", sizeof(con2.userName));
    printf("Size of phoneNumber : %lu\n", sizeof(con2.phoneNumber));
    printf("Size of globalNumber : %lu\n", sizeof(con2.globalNumber));
    printf("%s, +%d-%s\n", con1.userName, con1.globalNumber, con1.phoneNumber);
    printf("%s, +%d-%s\n", con2.userName, con2.globalNumber, con2.phoneNumber);
    return 0;
}
```

출력 예

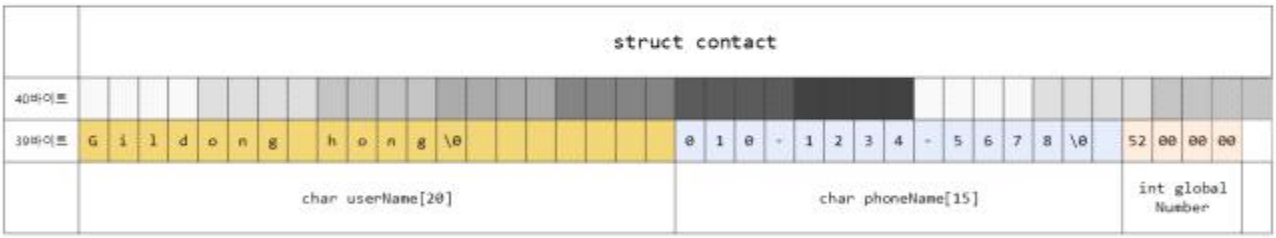
```
Size of struct contact : 40
Size of userName : 20
Size of phoneNumber : 15
Size of globalNumber : 4
Steve Jobs, +1-000-000-0000
Gildong Hong, +82-010-1234-5678
```


11.1 구조체

- 직접 멤버 연산자[ ]
  - ✓ 직접 멤버 [ ] 연산자
  - ✓ 구조체 변수 이름에 직접 멤버 연산자를 사용하여 멤버 변수에 [ ] 가능
  - ✓ 멤버 변수[ ]하기
    - ➢ (구조체 이름).(멤버 변수 이름)

11.1 구조체

- 구조체와 멤버 변수의 크기
  - ✓ struct contact의 크기 : ☐ 바이트
  - ✓ struct contact의 각 멤버 변수의 크기의 합 : ☐ 바이트



11.1 구조체

- 구조체와 멤버 변수의 크기
  - ✓ 구조체의 크기는 ☐ 바이트 단위로 할당
    - ➢ ☐ 의 배수 단위의 주소를 이용하여 메모리에 접근하는 것이 효율적
    - ➢ 구조체의 크기를 ☐ 의 배수 형태로 컴파일러가 맞춤
    - ➢ 구조체 변수를 선언하면 실제 멤버 변수에서 필요로 하는 크기보다 더 크게 ☐ 를 사용 가능

11.1 구조체

- 구조체의 멤버 변수
  - ✓ 변수, 배열, 포인터, 다른 [ ]를 멤버 변수로 선언 가능
  - ✓ 프로그램에서 사용하는 자료들 보다 더 [ ]할 수 있다는 장점

11.1 구조체

소스 11-2

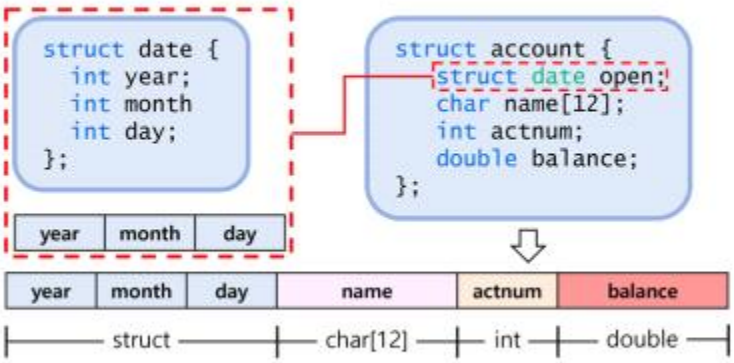
```c
#include <stdio.h>
struct date {
    int year;
    int month;
    int day;
};
struct account{
    struct date open;
    char name[12];
    int actnum;
    double balance;
};
int main(void) {
    struct account acc1 = {{2021, 02, 14}, "DateAccount", 4096, 1200000};
    printf("Size of struct account : %lu\n", sizeof(acc1));
    printf("[%d, %d, %d]\n", acc1.open.year, acc1.open.month, acc1.open.day);
    printf("%s, %d, %.2f\n", acc1.name, acc1.actnum, acc1.balance);
    return 0;
}
```

출력 예

```
Size of struct account : 40
[2021, 2, 14]
DateAccount, 4096, 1200000.00
```

11.1 구조체

- 구조체 [ ] 타입의 변수 메모리 할당



11.1 구조체

- 구조체 변수의 대입 방법
  - ✓ [ ] 구조체 타입 변수
    - ➢ 대입 연산자 [ ] 이용
    - ➢ 한 번에 모든 멤버 변수의 [ ] 변경 가능

11.1 구조체

소스 11-3

```c
#include <stdio.h>
struct date {
    int year;
    int month;
    int day;
};
int main(void) {
    struct date first_date = {2021, 9, 1};
    struct date second_date;
    second_date = first_date;
    printf("%d.%d.%d\n", second_date.year, second_date.month, second_date.day);
    return 0;
}
```

출력 예	2020.9.1

11.1 구조체

- 구조체 변수의 비교
  - ✓ [ ]의 멤버 변수로 구성 → 구조체 변수를 이용하여 [ ]는 불가능
  - ✓ 각각의 [ ]을 이용하여 비교

11.1 구조체

소스 11-4

```c
#include <stdio.h>
struct date {
int year;
int month;
int day;
};
int isEqualDate(struct date a, struct date b) {
if(a.year == b.year && a.month == b.month && a.day == b.day) return 1;
else return 0;
}
int main(void) {
struct date date1 = {2021, 11, 11};
struct date date2, date3 = {2021, 12, 25};
date2 = date1;
if(isEqualDate(date1, date2)) printf("Both are same.\n");
else printf("Both are not same.\n");
if(isEqualDate(date1, date3)) printf("Both are same.\n");
else printf("Both are not same.\n");
return 0;
}
```

출력 예

```
Both are same.
Both are not same.
```


11.1 구조체

- 구조체 [ ]과 [ ]
  - ✓ [ ]가 정의한 구조체
    - ➢ 프로그램 내에서는 하나의 데이터 타입으로 간주
    - ➢ 구조체를 이용한 배열이나 포인터의 사용도 가능

11.1 구조체

소스 11-5

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
struct studentInfo {
    char* univ;
    int sNo;
    char* major;
    char name[15];
};
int isEqualStudent(struct studentInfo a, struct studentInfo b) {
    if(!strcmp(a.univ, b.univ) &&!strcmp(a.major, b.major)
    &&!strcmp(a.name, b.name) && a.sNo==b.sNo) return 1;
    else return 0;
}
```


11.1 구조체

소스 11-4

```c
int main(void) {
    struct studentInfo student[4] =
                {{"DCU" , 21000001, "ComputerSW", "Kim"},
                {"DCU" , 21000002, "AIBigdata", "Lee"},
                {"DCU" , 21000003, "SWConvergence", "Park"}};
    int i;
    char temp[80];
    struct studentInfo *ptr =&student[3];
    printf("Input your name : ");
    scanf("%s", temp);
    strcpy(student[3].name, temp);
    printf("Input your student number : ");
    scanf("%d", &(ptr->sNo));
    printf("Input your major : ");
    scanf("%s", temp);
    ptr->major =malloc(sizeof(char)*(strlen(temp)+1));
    strcpy(ptr->major, temp);
    printf("Input your university : ");
    scanf("%s", temp);
    ptr->univ =malloc(sizeof(char)*strlen(temp));
    strcpy(ptr->univ, temp);
    for (i=0; i<4; i++) {
        printf("%s [%d, %s, %s]\n", student[i].univ,
            student[i].sNo,  student[i].major, student[i].name);
    }
    if(isEqualStudent(student[0], student[3]))
        printf("student 0 and student 3 are same.");
    else printf("student 0 and student 3 are not same.");
    return 0;
}
```

입력 예	Kim 21000001 ComputerSW DCU
출력 예	Input your name : Kim Input your student number : 21000001 Input your major : ComputerSW Input your university : DCU DCU [21000001, ComputerSW, Kim] DCU [21000002, AIBigdata, Lee] DCU [21000003, SWConvergence, Park] DCU [21000001, ComputerSW, Kim] student 0 and student 3 are same.

11.1 구조체

- 간접 멤버 연산자[ ]
  - ✓ 간접 멤버 [ ] 연산자
  - ✓ 구조체 포인터 변수를 이용하여 멤버 변수에 [ ]하기 위해 이용

- 구조체 [ ]에서 멤버 변수 접근 : 직접 멤버 연산자 (.)
- 구조체 [ ]에서 멤버 변수 접근 : 간접 멤버 연산자 (->)

11.2 공용체

- 공용체 [ ]
  - ✓ 동일한 [ ]에 여러 데이터타입을 저장할 수 있도록 하는 독특한 데이터 타입
  - ✓ 모든 멤버 변수가 공간을 [ ]하여 사용
  - ✓ 한 번에 [ ]가지 데이터 타입의 값만 저장 및 사용 가능
  - ✓ 공용체 사용 예
    - ➢ 데이터 [ ]을 특정 할 수 없는 경우
    - ➢ 기억 공간은 할당 받았지만, 어떤 데이터타입의 [ ]을 저장하는지 특정하지 못하는 경우

11.2 공용체

- 공용체의 정의 방법

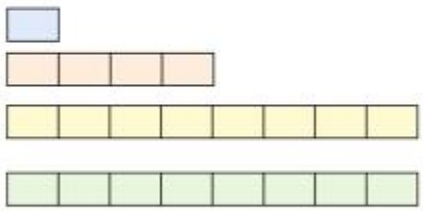
```
union value {
    char ch;
    int num;
    double fp;
};
```

char ch : 1 byte

int num : 4 byte

double fp : 8 byte

Size of union value : 8 byte



11.2 공용체

- 공용체의 정의 방법
  - ✓ union의 정의는 [ ]와 상당히 유사
  - ✓ union 키워드를 [ ] 키워드로 변경 시 바로 구조체가 됨
  - ✓ 공용체의 이름 : [ ]
  - ✓ [ ]를 정의해도 실제로 공용체는 메모리를 할당 받지 못함
    - ➢ 정의된 공용체를 이용하여 변수 선언 시, [ ] 할당

11.2 공용체

- 공용체의 정의 방법
  - ✓ [ ] 멤버 변수의 필요 공간만큼 공간 할당
  - ✓ 모든 멤버 변수가 값을 저장하고 [ ]
    - ➢ 공용체 변수의 [ ]된 값만 사용 가능
    - ➢ 그 데이터 타입에 따라 데이터를 저장
  - ✓ 모든 멤버 변수가 저장 공간을 [ ]하는 것 제외 시, 구조체와 동일
  - ✓ 공용체 변수를 이용하여 멤버 변수에 접근
    - ➢ [ ] 이용

11.2 공용체

소스 11-6

```c
#include <stdio.h>
union value {
    char ch;
    int n;
    double d;
};
int main(void) {
    union value v;
    printf("%lubyte\n", sizeof(v));
    v.ch ='F';
    printf("%c %lubyte\n", v.ch, sizeof(v.ch));
    v.n =0xff41;
    printf("%d %lubyte\n", v.n, sizeof(v.n));
    printf("%c %lubyte\n", v.ch, sizeof(v.ch));
    printf("%f %lubyte\n", v.d, sizeof(v.d));
    return 0;
}
```

출력 예

```
8byte
F 1byte
65345 4byte
A 1byte
0.000000 8byte
```

11.3 열거형

상수
- 리터럴 상수
- 심볼릭 상수
  - const 상수
  - 매크로 상수
  - 열거형(enumeration) 상수

11.3 열거형

- 열거형 [ ]
  - ✓ [ ] int형 값을 상수로 정의하는 방식의 상수
  - ✓ enum 키워드를 이용하여 정의
  - ✓ enum 뒤에 열거형 이름을 적고 [ ] 작성
  - ✓ 열거형 이름 : [ ]

11.3 열거형

- 열거형 [ ]
  - ✓ [ ] 안에 해당 열거형에 사용할 [ ] 이름 나열
    - ➢ 나열한 각각의 열거형 상수는 [ ] 값을 가짐
    - ➢ 별도의 값이 정의되지 않은 경우 [ ]에서부터 순서대로 [ ]씩 증가하는 값으로 정의
    - ➢ 열거형 상수에 [ ] 연산자를 이용하여 값 할당 시
      - 그 이후의 열거형 상수는 앞의 값에 [ ]을 더한 값으로 정의

11.3 열거형

- 열거형 [ ]
  - ✓ 여러 개의 [ ]를 정의해야 하는 경우
    - ➢ 매크로 상수나 [ ] 상수에 비해 편리하게 정의 가능
    - ➢ 소스 코드 가독성 상승
  - ✓ 상수 뿐만 아니라 [ ]도 선언 가능
  - ✓ 특정 열거형 변수
    - ➢ 해당 열거형 [ ]만을 값으로 가질 수 있는 변수
    - ➢ [ ] 않은 값은 가질 수 없는 변수

11.3 열거형

소스 11-7

```c
#include <stdio.h>
enum value { CV1, CV2, CV3 = 10, CV4};
int main(void) {
    printf("%d, %d, %d, %d\n", CV1, CV2, CV3, CV4);
    return 0;
}
```

출력 예	0, 1, 10, 11

11.3 열거형

소스 11-8

```c
#include <stdio.h>
enum LuxSkill {
    LightBinding = 1,
    PrismaticBarrier,
    LucentSingularity,
    FinalSpark
};
int main(void) {
    enum LuxSkill skill;
    char skill_btn;
    printf("Which skill do you want(q,w,e,r) >> ");
    scanf("%c", &skill_btn);
    if(skill_btn == 'q')skill = LightBinding;
    if(skill_btn == 'w')skill = PrismaticBarrier;
    if(skill_btn == 'e')skill = LucentSingularity;
    if(skill_btn == 'r')skill = FinalSpark;
    switch(skill){
    case LightBinding:
        printf("Light Binding\n");
        break;
    case PrismaticBarrier:
        printf("Prismatic Barrier\n");
        break;
    case LucentSingularity:
        printf("Lucent Singularity\n");
        break;
    case FinalSpark:
        printf("Final Spark\n");
        break;
    }
    return 0;
}
```

입력 예	r
출력 예	Which skill do you want(q, w, e, r) >> r Final Spark

11.3 열거형

- 열거형 활용
  - ✓ 요일 정의
    - ➢ 일요일에서부터 토요일까지의 요일을 0부터 6까지의 상수로 정의
    - ➢ 날짜를 7로 나눈 나머지와 비교하여 요일을 결정
  - ✓ 달 정의
    - ➢ 영어 단어 월을 열거형 상수로 정의

11.4 typedef

- [ ]
  - ✓ 데이터 타입 이름을 다른 이름의 데이터 타입으로 [ ]하는 키워드

ex) `typedef unsigned long ul;` 선언 시

`unsigned long i;` → `ul i;` 와 같이 선언 가능

  - ✓ [ ]한 데이터 타입은 [ ] 영역 내에서만 유효
    - ➢ 함수 [ ]에서 재정의 → 함수 안에서만 유효
    - ➢ 함수 [ ]에서 재정의 → 해당 소스 코드 파일 전체에서 유효
  - ✓ [ ]이름의 데이터 타입을 [ ]이름으로 재정의
  - ✓ 데이터 타입이 가지는 특정을 [ ]하게 알려주기 위한 용도로도 사용
    - ➢ 소스 코드 가독성 증가

11.4 typedef

소스 11-9

```
1  #include <stdio.h>
2  typedef unsigned int budget;
3  typedef int income, date;
4  budget getAnnualSalary(budget pay, date month) {
5      budget annual = pay * month;
6      return annual;
7  };
8  int main(void) {
9      date month =12;
10     income monthlyPay =3500000;
11     printf("Monthly pay is %d.\n", monthlyPay);
12     printf("Annual Salary is %d.\n", getAnnualSalary(monthlyPay, month));
13     return 0;
14 }
```

출력 예	MonthlyPay is 3500000. Annual Salary is 42000000.

11.4 typedef

- typedef
  - ✓ 하나의 데이터 타입을 여러 이름으로 [ ] 하는 경우
    - ➢ [ ]
  - ✓ [ ]된 데이터 타입
    - ➢ 함수의 반환값, 매개변수, 데이터 타입으로 사용 가능
    - ➢ 함수의 지역 변수 선언에도 사용 가능

11.4 typedef

- [ ] 재정의
  - ✓ [ ]의 경우, struct [ ]과 [ ]를 모두 적어 변수를 선언해야 하기에 typedef를 자주 사용

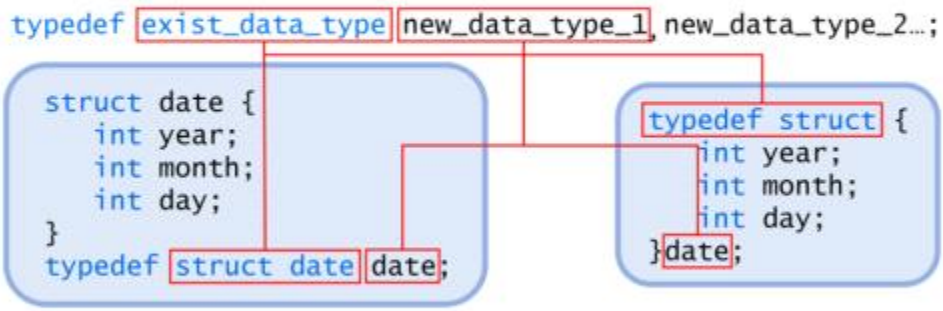
ex) typedef struct sturctName name; 선언 시

struct structName을 사용하지 않고 name으로 변수 선언 가능

  - ✓ 구조체 재정의 방식
    - ➢ [ ]과 동시에 typedef

11.4 typedef

- 구조체를 선언하고 별도로 typedef & 선언과 동시에 typedef 비교



11.4 typedef

소스 11-10

```c
#include <stdio.h>
struct yearMonthDay{
    int year;
    int month;
    int day;
};
typedef struct yymmdd{
    int yy;
    int mm;
    int dd;
} date2;
typedef struct yearMonthDay date1;
int main(void) {
    int y, m, d;
    date1 input;
    date2 ymd;
    printf("Input year, month, and day (ex.20 12 25) >> ");
    scanf("%d %d %d", &y, &m, &d);
    input.year = y;
    input.month = m;
    input.day = d;
    ymd.yy = y;
    ymd.mm = m;
    ymd.dd = d;
    printf("%d.%d.%d\n", input.year, input.month,
    input.day);
    printf("%d.%d.%d\n", ymd.yy, ymd.mm, ymd.dd);
    return 0;
}
```

입력 예	21 12 25
출력 예	Input year, month, and day (ex.20 12 25) >> 21 12 25 21.12.25 21.12.25

11.5 전처리기 활용과 매크로

- 전처리기 [ ]
  - ✓ C 언어로 작성한 소스 코드를 [ ] 전에 필요한 부분을 처리하기 위해 사용
    - ➢ 컴파일러가 소스 코드를 컴파일 하기 위해 필요한 [ ]을 수행
      - 소스 코드가 사용하는 [ ]을 소스 코드에 포함
      - 매크로 상수로 정의된 것을 [ ] 상수로 변경
  - ✓ 전처리기 지시자 [ ]
    - ➢ [ ]으로 시작하는 문장 ex) #include, #define 등
    - ➢ 전처리기 지시자를 통해 프로그래머가 소스 코드에 필요한 작업을 정의 가능

11.5 전처리기 활용과 매크로

- 전처리기 지시자의 종류 및 설명

전처리기 지시자	설명
#define	매크로 상수 또는 함수 정의
#elif	조건부 컴파일 블록 (else if 와 유사)
#else	조건부 컴파일 블록 (else와 유사)
#endif	조건부 컴파일 블록의 종료 표시
#if	주어진 연산식이 참이면 컴파일 (if와 유사)
#ifdef	주어진 이름이 정의되었다면 컴파일
#ifndef	주어진 이름이 정의되지 않았다면 컴파일
#include	지정된 헤더파일 내용을 현재에 복사
#undef	지정한 매크로 상수 또는 함수를 삭제

11.5 전처리기 활용과 매크로

- #if와 #endif
  - ✓ 기본적인 [ ] 컴파일 지시자
  - ✓ #if 다음의 조건식 expression이 0이 아니면
    - ➢ #endif 또는 다른 옵션의 전처리 지시자 사이의 [ ]을 컴파일
  - ✓ #if 다음의 조건식 expression이 0이면
    - ➢ 컴파일에서 제거
  - ✓ #if 사용 시, 주의점
    - ➢ [ ]를 만날 때까지 #if의 범위로 인식
    - ➢ [ ]를 사용하여 #if를 종료해야 함

11.5 전처리기 활용과 매크로

소스 11-11

```c
#include <stdio.h>
#define WINDOWS 1
#define MAC 2
#define UNIX 3
#define SYSTEM UNIX
#if (SYSTEM == WINDOWS)
typedef long cust_int;
#elif SYSTEM == MAC
typedef int cust_int;
#elif SYSTEM == UNIX
typedef long long cust_int;
#else
typedef short cust_int;
#endif
int main(void) {
    cust_int n = 17;
    printf("variable size : %d, save value : %d\n", sizeof(n), n);
    return 0;
}
```

출력 예	variable size : 8, save value : 17

11.5 전처리기 활용과 매크로

- [ ]
  - ✓ 매크로 상수가 정의되었을 경우
    - ➢ #ifdef과 #endif 사이의 모든 문장을 컴파일할 수 있도록 처리
  - ✓ 매크로 상수가 정의되지 않았을 경우
    - ➢ 컴파일에서 해당 부분의 문장들을 모두 제외
- [ ]
  - ✓ 매크로 상수가 정의되지 않았을 경우
    - ➢ #ifndef과 #endif 사이의 모든 문장을 컴파일 가능하도록 처리
  - ✓ 매크로 상수가 정의되었을 경우
    - ➢ 컴파일에서 해당 문장들을 제외
- [ ]
  - ✓ 이미 정의된 매크로 상수의 정의를 해제하는 지시자

11.5 전처리기 활용과 매크로

- 전처리기에 사용하는 연산자
  - ✓ #, #@, ##, define
  - ✓ 매크로 연산자의 종류 및 설명

연산자	이름	사용 예	기능
#	문자열 만들기 연산자 (Stringzing operator)	#인자	인자 앞 뒤에 큰 따옴표를 붙여 인자를 문자열로 만드는 여산자
#@	문자 만들기 연산자 (charizing operator)	#@인자	인자 앞 뒤에 작은 따옴표를 붙여 인자를 문자로 만드는 연산자
##	토큰 붙이기 연산자 (token-pasting operator)	인자##인자	인자를 다른 노큰들과 연결해주는 연산자
define	정의 검사 연산자 (defined operator)	defined WINDOWS	상수로 정의되어 있는지 검사하는 연산자

11.5 전처리기 활용과 매크로

- 연산자 - #, #@, ##
  - ✓ 매크로 정의 지시자 [ ] 에서 사용
- [ ]
  - ✓ 조건부 컴파일 지시자 [ ], [ ] 등과 함께 사용 가능
- 문자열 만들기 연산자 - [ ]
  - ✓ 매크로 함수 또는 상수 정의 시, 뒤이어 나오는 인자를 [ ] 로 만드는 연산자
  - ✓ 매크로 함수에서 인자를 [ ] 에 넣어 문자열로 변경

11.5 전처리기 활용과 매크로

- 문자 만들기 연산자 - [ ]
  - ✓ 매크로 정의에서 형식인자 앞에 위치 (인자 x → [ ])
  - ✓ 뒤에 나오는 인자를 [ ]를 붙여 문자로 변경
- 토큰 붙이기 연산자 - [ ]
  - ✓ [ ] 토큰을 연결

11.5 전처리기 활용과 매크로

- 예약 매크로 [ ]
  - ✓ 사전에 시스템에 의해 정의된 [ ]
  - ✓ C 언어에서는 프로그램 편의를 위해 시스템에 사전에 정의된 예약 [ ] [ ]를 제공
  - ✓ [ ] 표준 매크로 프로그램 디버깅에 활용 가능

11.5 전처리기 활용과 매크로

- 사용 빈도가 비교적 높은 예약 매크로

매크로	설명
__DATE__	가장 최근에 소스 파일을 컴파일한 날짜를 [Mmm dd yyyy]으로 표시
__FILE__	현재 소스 파일 이름으로 절대 경로로 표시
__LINE__	현재 소스 파일에서 이 문장이 있는 줄 번호
__TIME__	가장 최근에 소스 파일을 컴파일한 시각으로 [시:분:초]로 표시
__TIMESTAMP__	현재 소스 파일을 마지막 수정한 시각으로 [요일 월 날짜 시:분:초 년]으로 표시

11.5 전처리기 활용과 매크로

소스 11-12

```c
#include <stdio.h>
int main(void) {
    printf("__DATE__ -> %s\n", __DATE__);
    printf("__FILE__ -> %s\n", __FILE__);
    printf("__LINE__ -> %d\n", __LINE__);
    printf("__TIME__ -> %s\n", __TIME__);
    printf("__TIMESTAMP__ -> %s\n", __TIMESTAMP__);
    return 0;
}
```

출력 예

```
__DATE__ -> Nov 3 2020
__FILE__ -> main.c
__LINE__ -> 5
__TIME__ -> 03:49:08
__TIMESTAMP__ -> Tue Nov 3 03:49:07 2020
```

11.6 구조체 응용

- 연결 리스트 [ ]
  - ✓ [ ]으로 데이터가 할당되는 형태의 자료구조
  - ✓ 배열과 유사한 [ ] 구조
    - ➢ 임의의 원소에 [ ]이 불가능한 구조 (배열과 차이점)
  - ✓ 제일 앞 또는 뒤에서부터 [ ]으로 [ ]씩 데이터를 읽는 형태
  - ✓ [ ]을 효율적으로 사용 가능
  - ✓ 자주 사용되는 자료 구조

11.6 구조체 응용

- 연결 리스트 [ ]
  - ✓ 데이터를 [ ]하는 구성 요소
    - ➢ 배열 : [ ]
    - ➢ 연결 리스트 : [ ]
      - 제일 앞에 위치한 노드를 가리키는 포인터 : [ ]
      - 제일 마지막에 위치한 노드를 가리키는 포인터 : [ ]
  - ✓ [ ]와 [ ]을 기준으로 [ ]으로 자료에 접근
  - ✓ [ ]의 추가 및 삭제가 용이
  - ✓ [ ]으로 메모리 관리 → [ ]의 수 제한 없음

11.6 구조체 응용

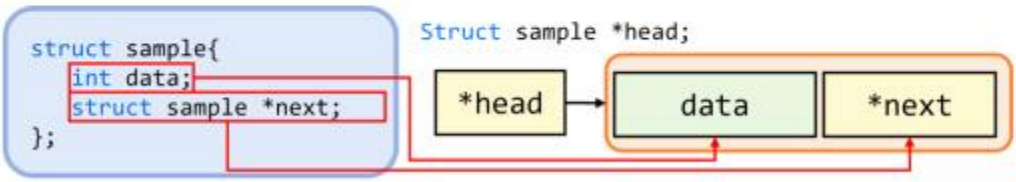
- 연결 리스트와 배열
  - ✓ 중간에 위치한 원소를 삭제할 경우
    - ➢ 배열 : 뒤에 있는 모든 원소를 [ ] 칸씩 이동
    - ➢ 연결리스트 : 추가 작업 없음
  - ✓ 데이터 삽입할 경우
    - ➢ 배열 : 삽입하고자 하는 위치 [ ]의 모든 원소를 [ ] 칸씩 뒤로 이동
    - ➢ 연결리스트 : 추가 작업 없음
  - ✓ [ ]에 접근 방식
    - ➢ 배열 : 원소의 [ ]를 이용
    - ➢ 연결리스트 : 헤드 또는 테일로부터 각 노드를 [ ]으로 접근

11.6 구조체 응용

- 연결 리스트 [ ]
  - ✓ 각 노드는 자신의 다음 노드에 대한 [ ]를 가짐 → [ ] 접근
  - ✓ 연결리스트의 종류
    - ➢ [ ]으로만 접근 가능한 연결리스트 (헤드 → 테일)
    - ➢ [ ] 연결리스트
      - [ ] 모두 순회가 가능한 연결리스트 (헤드 → 테일, 테일 → 헤드)
      - 각 노드는 이전 노드에 대한 [ ]와 다음 노드에 대한 [ ] 모두 가짐

11.6 구조체 응용

- 연결 리스트 [ ]
  - ✓ 구조체를 이용하여 구현
  - ✓ 구조체의 멤버 변수로 자기 자신에 대한 포인터 변수를 가짐
  - ✓ 연결리스트 구조



11.6 구조체 응용

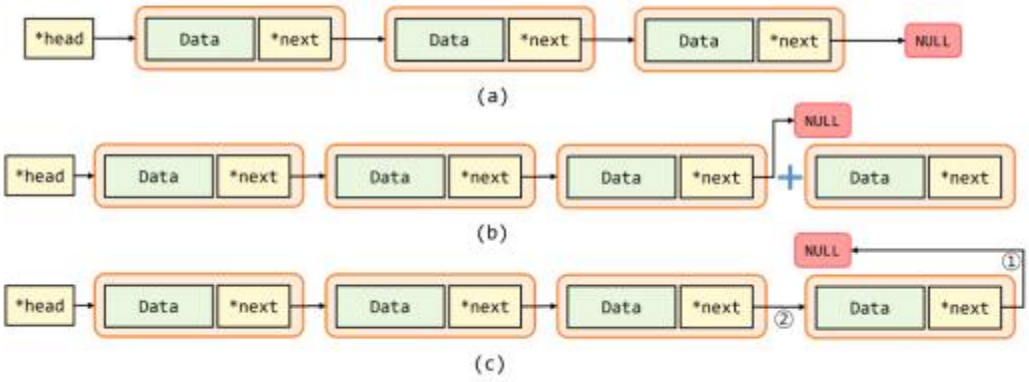
- 연결 리스트 [ ]
  - ✓ 프로그램 실행 전, 미리 공간 확보할 필요 없음
  - ✓ 프로그램 실행 중, 필요할 때 노드를 동적으로 추가 및 삭제
  - ✓ 포인터를 이용하여 순차적 접근
    - ➢ 임의로 노드에 접근 → 배열에 비해 많은 시간 소요

11.6 구조체 응용

- 연결 리스트 노드 [ ]
  - ✓ 모든 노드를 [ ] 검색하는 방법
  - ✓ [ ]를 기점으로 포인터를 이용해 마지막 노드의 링크가 [ ]이 될 때까지 [ ]

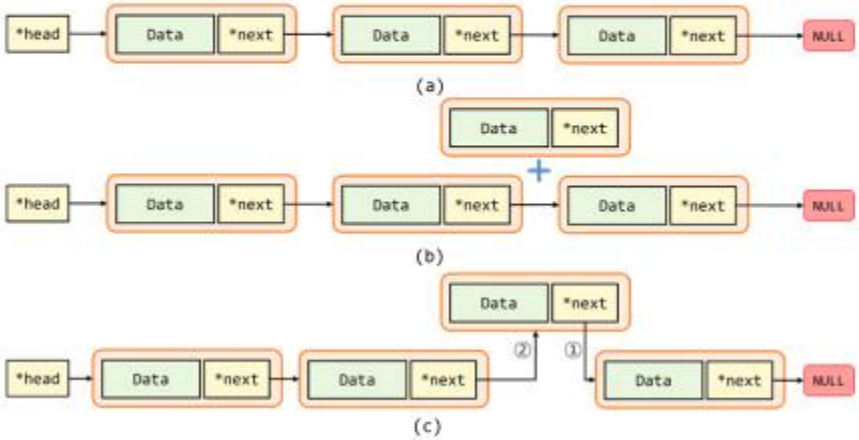
11.6 구조체 응용

- 연결 리스트 노드 [ ]
  - ✓ 연결리스트 [ ]에 새로운 노드를 [ ]하는 과정



11.6 구조체 응용

- 연결 리스트 노드 [ ]
  - ✓ 연결리스트 [ ]에 새로운 노드를 [ ]하는 과정

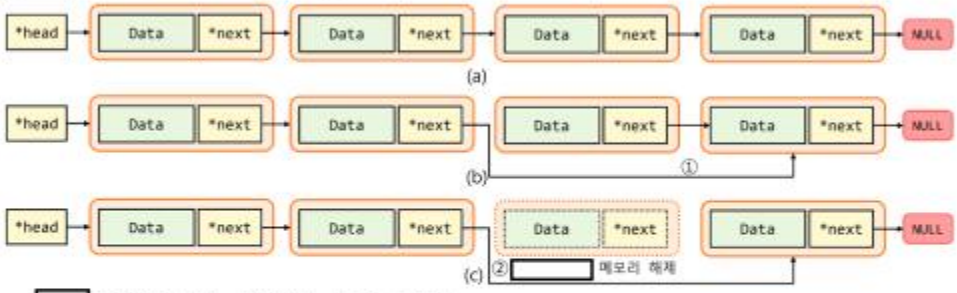


11.6 구조체 응용

- 연결 리스트 노드 [ ]
  - ✓ [ ] 삽입과 [ ] 삽입은 동일한 동작 과정
  - ✓ 삽입 [ ] 노드의 포인터가 가리키는 것을 새로운 노드가 가리킴
    - ➢ [ ] 삽입 : NULL
    - ➢ [ ] 삽입 : 다음 노드
  - ✓ [ ]의 포인터가 새로운 노드를 가리킴

11.6 구조체 응용

- 연결 리스트 노드 [ ]



1. [ ]하고자 하는 타겟 노드를 설정
2. [ ] 노드 포인터를 타깃 노드가 가리키는 노드로 설정
3. [ ] 함수를 이용해 노드를 메모리에서 해제하여 삭제
4. 타깃 노드는 연결이 [ ] 되었음으로 사용 불가능

11.6 구조체 응용

소스 11-13

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct LinkedList{
    char* name;
    struct LinkedList* next;
};
typedef struct LinkedList NODE;
typedef NODE* LINK;
LINK createNode(char* name);
LINK append(LINK head, LINK curr);
int printList(LINK head);
LINK createNode(char* name) {
    LINK curr;
    curr = (LINK)malloc(sizeof(NODE));
    if(curr ==NULL){
        printf("Memory allocation problem!\n");
        return NULL;
    }
    curr->name = (char*)malloc(sizeof(char) * (strlen(name) +1));
    strcpy(curr->name, name);
    curr->next =NULL;
    return curr;
}
LINK append(LINK head, LINK curr) {
    LINK nextNode = head;
    if(head ==NULL){
        head = curr;
        return head;
    }
    while(nextNode->next !=NULL) {
        nextNode = nextNode->next;
    }
    nextNode->next = curr;
    return head;
}
int printList(LINK head){
    int cnt =0;
    LINK nextNode = head;
    while(nextNode !=NULL) {
        printf("Node #%d is %s\n", ++cnt, nextNode->name);
        nextNode = nextNode->next;
    }
    return cnt;
}
```

11.6 구조체 응용

소스 11-13

```
46 int main(void){
47     char name[30];
48     LINK head =NULL;
49     LINK curr;
50     printf("Input data >> ");
51     while(scanf("%s", name) !=NULL) {
52         curr = createNode(name);
53         if(curr ==NULL) {
54             printf("Memory allocation problem!");
55             return -1;
56         }
57         head = append(head, curr);
58         printList(head);
59         printf("Input data >> ");
60     }
61     return 0;
62 }
```

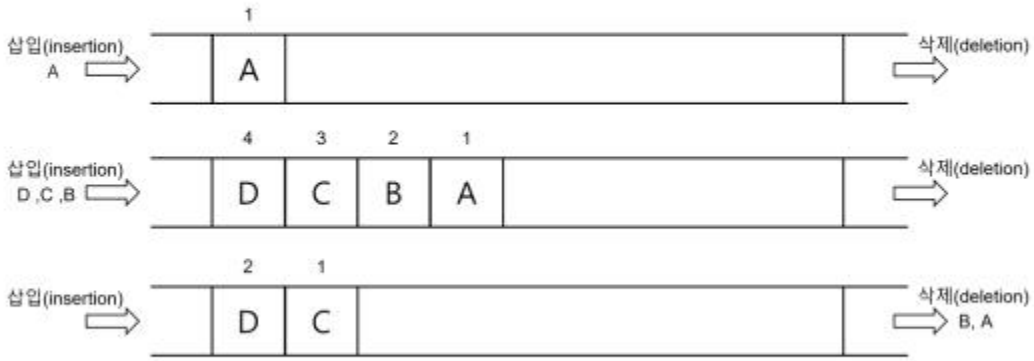
입력 예	DCU Computer_Science Song
출력 예	Input the data >> DCU Node #1 is DCU Input the data >> Computer_Science Node #1 is DCU Node #2 is Computer_Science Input the data >> Song Node #1 is DCU Node #2 is Computer_Science Node #3 is Song Input the data >>

11.6 구조체 응용

- 큐 [ ]
  - ✓ [ ]를 이용한 자료구조
  - ✓ 가장 일반적
  - ✓ [ ] 입력된 값이 [ ] 출력되는 선형구조
  - ✓ 선입선출[ ]
  - ✓ 배열, 연결리스트로 구현 가능

11.6 구조체 응용

- 큐의 기본 동작 구조



11.6 구조체 응용

소스 11-14

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct Node {
    int data;
    struct Node* next;
}Node;
typedef struct Queue {
    Node*front;
    Node* rear;
    int count;
}Queue;
void InitQueue(Queue*queue);
int IsEmpty(Queue*queue);
void Enqueue(Queue*queue, int data);
int Dequeue(Queue*queue);
```

```
void InitQueue(Queue*queue) {
    queue->front =queue->rear =NULL;
    queue->count =0;
}
int IsEmpty(Queue*queue) {
    return !queue->count;
}
void Enqueue(Queue*queue, int data) {
    Node* now = (Node*)malloc(sizeof(Node));
    now->data = data;
    now->next =NULL;
    if (IsEmpty(queue)) {
        queue->front = now;
    }
    else {
        queue->rear->next = now;
    }
    queue->rear = now;
    queue->count++;
}
```

11.6 구조체 응용

소스 11-13

```
37 int Dequeue(Queue*queue) {
38     int re =0;
39     Node* now;
40     if (IsEmpty(queue)) {
41         return re;
42     }
43     now =queue->front;
44     re = now->data;
45     queue->front = now->next;
46     free(now);
47     queue->count--;
48     return re;
49 }
```

```
50 int main(void) {
51     int i;
52     Queue queue;
53     InitQueue(&queue);
54     for (i =1; i <=5; i++) {
55         Enqueue(&queue, i);
56     }
57     while (!IsEmpty(&queue)) {
58         printf("%d ", Dequeue(&queue));
59     }
60 printf("\n");
61 return 0;
62 }
```

출력 예	1 2 3 4 5

11.6 Q&A

Q 45. 동적으로 할당한 어떤 오브젝트를 가리키는 포인터를 포함하는 구조체를 할당했다. 이 구조체를 해제할 때, 각각의 포인터 멤버가 가리키는 메모리도 따로 해제해 주어야 하는가?

A. 그렇다. malloc()으로 할당한 메모리는 각각, 정확히 딱 한번, free() 해주어야 한다. 프로그램에서 각각 malloc()으로 할당한 메모리는 모두 free()시키는 것이 좋은 습관이다.

11.6 Q&A

Q 46. 이 프로그램은 정상적으로 동작하지만 프로그램이 끝났을 때, core파일을 만들어 낸다. 왜 그런가?

```
struct list{
  char *item;
  struct list **next;
}
main(argc, argv)
{...}
```

A . 구조체를 정의할 때 세미콜론(';')을 빠뜨렸기 때문에 main()이 구조체를 리턴하는 함수로 정의되어 버렸다. (중간에 들어간 주석(comment) 때문에 이 버그를 찾아내기가 더욱 힘들 것이다.). 대부분 구조체를 리턴하는 함수들은 숨겨진 리턴 포인터(hidden return pointer)를 추가하는 식으로 구현되기 때문에, main()이 3개의 인자를 받는 것처럼 만들어진다. 원래 C start up code는 main()이 두 개의 인자를 받는 것으로 작성되어 있으므로, 이 경우 정상적으로 동작할 수 없다.

8.7 Q&A

Q 47. 구조체 타입에 sizeof 연산자를 썼더니, 예상한 것 보다 훨씬 큰 값을 리턴한다. 왜 그런가?

A. 구조체는 필요한 경우 이러한 'padding' 공간을 포함할 수 있다. 이는 구조체가 배열로 만들어질 때, 정렬(alignment) 속성이 보존되도록 하기 위한 것이다. 또 배열로 쓰이지 않을 경우에도 이러한 여분의 'padding'이 남아 있을 수 있다. 이는 sizeof가 일관된 크기를 리턴하기 위한 것이다.

1.7 실습 및 과제

- 실습 및 과제 진행
  - DCU Code : http://code.cu.ac.kr
  - 11 주차 실습
    - 코딩 11-1, 코딩 11-2, 코딩 11-3, 코딩 11-4, 코딩 11-5, 코딩 11-6, 코딩 11-7 (p412-447)
  - 11 주차 과제
    - 11장 프로그래밍 연습 1-4번 (p454)

E N D