실습1

|  |
| --- |
| #include<stdio.h>  typedef struct {  char Name[100];  int age;  int digit;  } Student;  void swap(Student\* a, Student\* b){  Student c;  c = \*a;  \*a = \*b;  \*b = c;  }  void PRINT(const Student x) {  printf("Name : %s\n", x.Name);  printf("age : %d\n", x.age);  printf("digit : %d\n\n", x.digit);  }  void PRINTINFO(const Student \*x) {  printf("Name : %s\n", x->Name);  printf("age : %d\n", x->age);  printf("digit : %d\n\n", x->digit);  }  int main() {  Student a = { "Miru", 3, 191128 };  Student b = { "Ruri", 3, 200201 };  PRINT(a);  PRINT(b);  swap(&a, &b);  PRINTINFO(&a);  PRINTINFO(&b);  } |

텍스트이(가) 표시된 사진

자동 생성된 설명

친구관리

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #include<stdio.h>  #include<stdlib.h>  #include <io.h>  typedef struct \_friend {  char\* name;  char\* message;  int status;  } friend;  friend\* list[100];  int list\_num;  void Add() {  int size = 0;  int mes = 0;  friend\* frd = 0;  frd = (friend\*)malloc(sizeof(friend));  printf("name size: ");  scanf("%d", &size);  frd->name = (char\*)malloc(size);  printf("name: ");  int len;  len = \_read(0, frd->name, size);  if (frd->name[len - 1] == '\n')  frd->name[len - 1] = '\0';  printf("message size: ");  scanf("%d", &mes);  frd->message = (char\*)malloc(mes);  printf("message: ");  int lenn;  lenn = \_read(0, frd->message, mes);  if (frd->message[lenn - 1] == '\n')  frd->message[lenn - 1] = '\0';  printf("status (0 / 1/ 2) : ");  scanf("%d", &frd->status);  list[list\_num++] = frd;  }  void Delete() {  int del\_num;  printf("몇번째 학생을 삭제하실 건가요? : ");  scanf("%d", &del\_num);  for (int i = del\_num - 1; i < list\_num - 1; i++) {  memcpy(&list[i], &list[i + 1], sizeof(friend));  }  }  void Print\_one() {  char find[10];  printf("검색할 이름을 입력해주세요 : ");  scanf("%s", find);    for (int i = 0; i < list\_num; i++) {  if (strcmp(list[i]->name, find)==0) {  printf("name: %s\n", list[i]->name);  printf("message: %s\n", list[i]->message);  printf("status: %d\n", list[i]->status);  }    }  }  void View() {    printf("======== List =========\n");  for (int i = 0; i < list\_num; i++) {  printf("name: %s\n", list[i]->name);  printf("message: %s\n", list[i]->message);  printf("status: %d\n", list[i]->status);  printf("------------------------------\n");  }  }  void Bye() {  printf("프로그램을 종료하겠습니다.\n");  return 0;  }  int main() {  printf("======== CHOOSE NUMBER =========\n");  printf(" 1. Add \n");  printf(" 2. Delete \n");  printf(" 3. Print\_one \n");  printf(" 4. View \n");  printf(" 5. Bye \n");  int num = 0;  while (num != 5) {  printf("번호를 입력하세요 : ");  scanf("%d", &num);  switch (num) {  case 1:  Add();  break;  case 2:  Delete();  break;  case 3:  Print\_one();  break;  case 4:  View();  break;  case 5:  Bye();  break;  }  }  return 0;  } |

텍스트이(가) 표시된 사진

자동 생성된 설명

**#1 calloc, realloc 조사하고 실습**

**Calloc**

함수원형 : void\* calloc(size\_t elt\_count, size\_t elt\_size)  
  
매개 변수를 2개 받아서 둘을 곱한 만큼의 메모리 공간을 반환함

Malloc 과 가장 큰 차이 : 0으로 초기화가 이루어졌는가?

+ 메모리 사이즈를 받는 파라미터의 개수가 다름

|  |  |  |
| --- | --- | --- |
|  | malloc | Calloc |
| 초기화 | 처음에 쓰레기값 | 처음에 0 (초기화O) |
| 블록수 | 요청된 메모리의 단일 블록을 할당 | 요청된 메모리의 여러 블록을 할당 |
| 속도 | 빠름 | 비교적 천천히 |

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  int main() {  int\* pi1;  int\* pi2;  pi1 = (int\*)malloc(sizeof(int) \* 3);  printf("malloc : %d %d %d\n", pi1[0], pi1[1], pi1[2]);  pi2 = (int\*)calloc(3, sizeof(int));  printf("calloc : %d %d %d\n", pi2[0], pi2[1], pi2[2]);  free(pi1);  free(pi2);  } |

텍스트이(가) 표시된 사진

자동 생성된 설명

사이즈 3을 할당했을 때 malloc은 쓰레기값, calloc 은 0으로 초기화된 것을 확인

**Realloc** : 동적 할당된 메모리의 내용을 유지하면서 할당된 메모리의 크기를 변경해주는 함수

함수원형 : void\* realloc(void \*ptr,size\_t size)

Ptr: 메모리 크기를 변경시킬 포인터

Size: 할당받을 메모리 크기

사용자가 요구한 크기만큼 연속되게 메모리를 확보하지 못함

→ 해당 주소를 버리고 다른 공간에서 필요한 만큼 메모리를 확보함

주소만 달라질 뿐, 해당 메모리가 가지고 있던 내용은 유지

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  int main() {    char\* arr = (char\*)malloc(sizeof(char) \* 3);    arr = (char\*)realloc(arr, sizeof(char) \* 10);  } |

길이를 3에서 10인 배열로 늘려줌

**#2 공용체(union)에 대해 조사 & 간단한 실습**

**공용체 :** 사용자가 정의한 자료형

**+ 구조체 :** 하나 이상의 서로 다른 종류의 변수들을 묶어서 새로운 자료형을 정의하는 것

(구조체는 각 변수가 각자의 메모리를 차지, 공용체는 메모리를 공유)

공용체의 메모리는 선언된 것 중 가장 큰 바이트로 할당됨

공용체로 정의한 변수는 하나의 메모리 공간을 여러 방식으로 접근할 수 있음

→ 메모리는 공유하고 싶은데 다르게 해석하고 싶을 때 사용

**구조체** – 멤버 변수가 각각의 메모리 공간을 할당받고 있음

**공용체** – 같은 메모리공간을 모든 변수가 공유하는 형태

공체로 메모리 절약 가능, But 공용체 멤버가 동시에 사용하면 데이터가 변질될 수 있으니 따로 사용해야함

|  |
| --- |
| union 공용체명  {     멤버1     멤버2  }; |



|  |
| --- |
| #include <stdio.h>  union data  {  int age;  char name;  };  int main(void)  {  union data cat;  cat.age = 18;  cat.name = "민영";  printf("age : %d \n", cat.age);  printf("score : %d \n", cat.name);  printf("cat의 크기 : %d \n", sizeof(cat));  return 0;  } |

텍스트이(가) 표시된 사진

자동 생성된 설명

**#3 열거형(enum)에 대해 조사**

**Enum :** 변수를 나열해서 사용

선언방법 : enum 변수명{내용};

√ 구조체나 공용체는 중괄호 안에 변수 선언, 열거형은 콤마로 구분자를 넣어 문자를 지정

√ 열거형으로 선언된 순서에 따라 정수 0부터 index 값을 지님

Eun 변수명 {a=2, b=3, c}; 형식으로 원하는 값을 지정해서 정의할 수 있음

√ 값을 지정하지 않으면 자동적으로 +1의 값을 지님

√ 클래스 변수와 마찬가지로 바로 사용 가능

√ 열거형 타입에는 열거형 상수와 null 값 할당 가능

√ 프로그램의 가독성을 높여줌

√ 접근 제한자는 public 과 default 만 가능

+) 접근 제한자 public과 default?

Public : 외부의 모든 클래스에서 접근 가능함

Default : private + 같은 패키지 안에서 접근 가능

Private : 속해있는 class 에서만 접근 가능

**#4 Socket의 개념(종류, 역할, 용도 등 자세히)과**

**Socket 통신(TCP 프로토콜 통신)의 과정 조사**

**Socket:** 프로세스가 네트워크를 통해 데이터를 송수신 할 수 있도록 하는 창구,

떨어져있는 두 호스트를 연결해주는 인터페이스 역할을 함

프로토콜, IP 주소, 포트 넘버로 정의

역할) 서버 소켓, 클라이언트 소켓

**서버소켓**

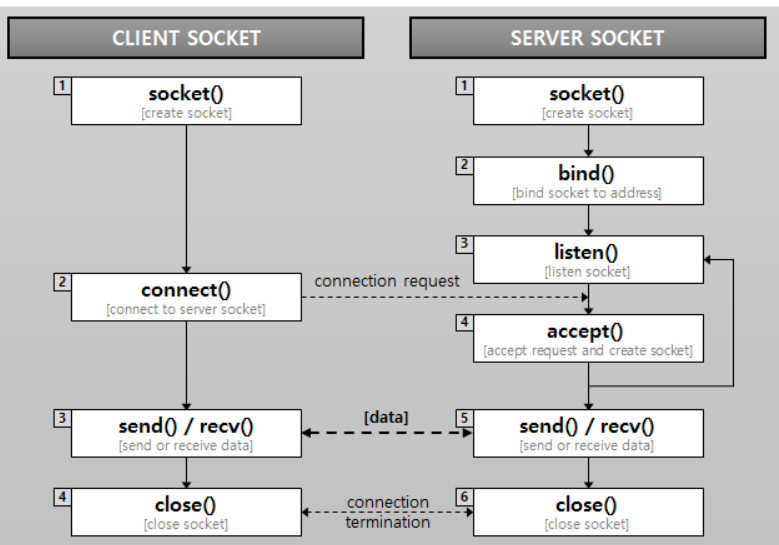
서버 프로그램에서만 사용하는 소켓

클라이언트로부터 연결 요청을 기다렸다가 연결 요청이 들어오면 클라이언트와 연결을 맺고 다른 소켓을 만듦

사용방법: Sever socket =

클라이언트 소켓

**흐름**



1. Socket() 함수를 이용해서 소켓 생성
2. Bind() 함수로 ip와 port 번호를 설정
3. Listen() 함수로 클라이언트의 접근 요청에 수신 대기열을 만듦 -> 몇 개의 클라이언틀르 대기 시킬지 결정
4. Accept() 함수를 사용하여 클라이언트와의 연결을 기다림

**소켓 종류**

**스트림 (TCP)**

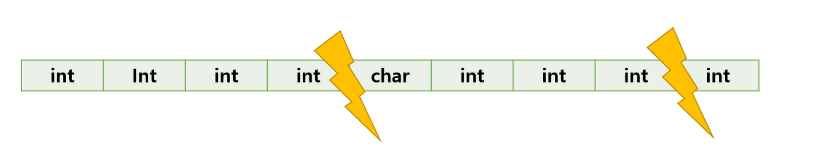
* 양방향으로 바이트 스트림을 전송, 연결 지향성
* 오류 수정, 흐름 제어 보장
* 송신된 순서에 따라 중복 X 데이터를 수신,
* 소량의 데이터보다는 대량의 데이터 전송에 적합 -> TCP를 사용

**데이터그램 (UDP)**

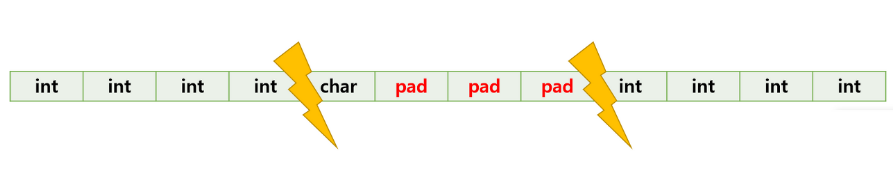
* 비연결형 소켓
* 데이터의 크기에 제한 O
* 확실하게 전달이 보장 X, 데이터가 손실돼도 오류 발생 X
* 실시간 멀티미디어 정보를 처리하기 위해 주로 사용

**#5 why?**

**-구조체 패딩-**



5byte 선언해주면 마지막 1byte가 남아서 연산을 두 번 처리하게 됨



구조체 패딩을 통해 8byte를 선언해주고 나머지 부분에는 패딩 바이트가 들어가게 함

* Cpu는 한 번의 동작만 하면 되어 연산이 보다 빠르게 처리가됨