C 스터디

-컴파일러

컴파일러는 프로그래밍 언어(고급 언어)로 작성한 프로그램을 컴퓨터가 이해할 수 있도록 기계어로 번역하는 역할을 한다.

\*컴파일언어와 인터프리터언어의 차이점

컴파일 언어는 컴파일러로 전체를 번역하므로, 번역 시간이 오래 걸리지만, 한 번 번역한 후에는 다시 번역하지 않아 실행 속도가 빠르다. 대표적인 컴파일 언어로는 C, C++ 등이 있다. 인터프리터 언어는 고급언어로 작성된 프로그램을 한 줄씩 읽어 들여서 실행하는 방식으로 번역과 실행이 동시에 이루어져 번역 속도는 빠르지만 실행 속도는 느리며 별도의 실행 파일이 존재하지 않는다. 대표적으로 R, Python, javascript 등이 있다. 코드 수정 시 인터프리터 언어는 별도의 빌드 과정 없이 수정할 수 있으나 컴파일 언어는 빌드 과정 전체를 다시 거쳐야 한다.

-디버그모드와 릴리즈모드의 차이

디버그모드

실행파일에 디버깅 정보를 삽입하여 언제든지 디버깅을 할 수 있도록 하며 Debug 서브 폴더에 실행파일을 만들어준다. 디버깅 정보가 들어가 있어 실행파일 상태를 확인할 수 있다. 디버그에 필요한 정보들을 실행 시 계속 체크해 속도가 느리다.

릴리즈모드

디버깅 정보를 삽입하지 않고 코드를 최적화하여 실행 파일 크기를 최대한 줄여준다. 따라서 속도나 크기면에서 디버그 모드보다 유리하며 현재 버전에서 결함이나 문제를 발견할 수 없을 때 릴리즈 모드로 컴파일한다.

-리틀엔디안 VS 빅엔디안

엔디안

컴퓨터의 메모리와 같은 1차원 공간에 여러 개의 연속된 대상을 배열하는 방법

작은 단위가 앞에 나오는 리틀 엔디안 : 최하위 바이트부터 차례로 저장하는 방식, 메모리에 저장된 값의 하위 바이트들만 사용할 때 별도의 계산이 필요 없다.

큰 단위가 앞에 나오는 빅 엔디안 : 최상위 바이트부터 차례로 저장하는 방식, 사람이 숫자를 읽고 쓰는 방법과 같아 디버깅 과정에서 메모리의 값을 보기 편하다.

\*0x41424344가 빅엔디안일 때 리틀엔디안으로는? -> 0x44434241

-.c파일이 실행파일이 되기까지의 과정

1. preprocessing (전처리 과정)

전처리 구문을 처리하는 과정으로 이미 정해져 있는 것들을 먼저 처리하여 이후에 다시 찾아 쓸 필요 없이 정리해두는 과정이다. ex) #include <stdio.h>

.c 파일을 입력-> .i 확장자 파일을 생성

2. compilation (컴파일)

고급 언어를 어셈블리어 언어로 변환하는 과정으로 기계어와 가까운 형태의 언어로 번역하는 과정이다.

.i 파일 -> .s 파일

3. assemble (어셈블)

어셈블리어를 기계어로 번역하는 과정, 목적 파일을 생성한다.

.s 파일 -> .o 파일

4. linking (링킹)

각 파일들이 기계어로 번역된 후 하나로 연결하는 과정으로 여러 개의 목적 파일을 합치거나 라이브러리를 합친다.

.o 파일 -> .exe 실행파일

-기계어와 어셈블리어

기계어는 컴퓨터가 사용하는 언어로 0과 1로 구성된 2진수로 이루어져 있다.

어셈블리어는 기계어 조합을 간단하게 이해할 수 있는 영단어로 대체한 것인 니모닉으로 이루어진 언어이다. 어셈블리어의 경우 제조사가 다르면 니모닉을 구성하는 이진수 조합도 다르기 때문에 호환성이 떨어진다.

기계어-어셈블리어-고급 언어(컴파일러 언어, 인터프리터 언어)-사람의 언어

하위 언어는 컴퓨터가 이해하기 쉽고 상위 언어로 갈 수록 사용자가 이해하기 편하다. 따라서 기계어로 번역해 명령하는 과정이 필요하다.

-난수 관련 함수 조사, 실습

rand() : 숫자를 무작위로 난수를 출력해주는 함수

0부터 RAND\_MAX(32767)까지의 정수를 생성

srand() : 호출할 때 전달받은 인자(시드)를 기반으로 난수를 초기화 해주는 함수

time()을 이용하여 시간 값을 매개로 초기화 하면 일정하지 않고 불규칙적인 난수를 생성할 수 있다.

time() : UTC기준 1970년 1월 1일 0시부터 흐른 시간을 초로 반환해주는 함수

C언어에서 난수 함수를 사용하기 위해서는 stdlib.h, time.h 헤더파일을 선언해 주어야한다. stdlib.h헤더파일에는 rand, srand 함수가 정의되어 있고 time.h 헤더파일에는 time함수가 정의되어 있다.

\*헤더파일 : 함수 원형들을 미리 선언해 두는 파일, 확장자 .h

헤더파일의 선언은 소스파일의 맨 앞부분, main 함수 정의 이전에 와야한다.

대표적 전처리기에는 #include가 있고 컴파일러에게 안내된 파일을 읽게 한다.

time함수로 구한 초를 기반으로 srand 함수에 난수 기준값을 초기화하고 rand함수를 호출하면 난수가 생성된다.

(rand( ) % (b-a+1)) + a --> [a , b] 구간의 난수 생성

소스코드

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  int main() {  int rnd,quiz;  srand(time(NULL));  rnd = (rand() % 5) + 1; //1~5  do {  printf("1~5 사이 숫자를 맞춰보세요:");  scanf("%d", &quiz);  } while (quiz != rnd);  printf("정답입니다.");  } |

출력 결과

텍스트이(가) 표시된 사진

자동 생성된 설명

- call by reference & call by value 조사 및 구현

call by reference 참조에 의한 호출 : 메모리의 접근에 사용되는 주소 값을 전달하는 형태의 함수호출, 변수의 주소 값을 인자로 받아서 해당 변수에 직접 접근하는 형태를 띤다.

장점 : 복사하지 않고 직접 참조를 하기에 실행속도가 빠르다.

단점 : 직접 참조를 하기에 원래 값이 영향을 받는다. 원래 값에 대한 즉각적인 수정 작업이기에 위험한 측면이 있다.

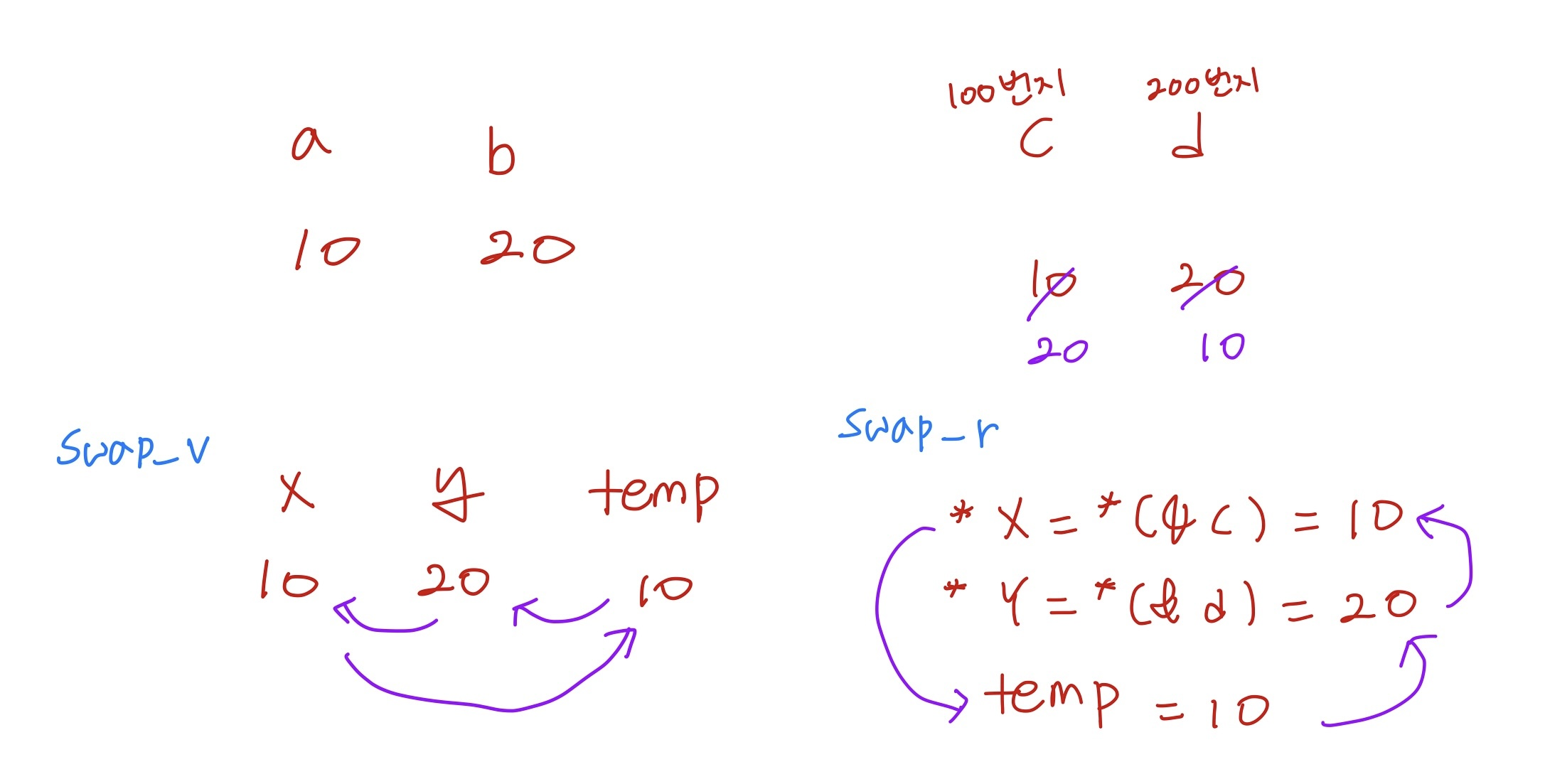
call by value 값에 의한 호출 : 단순히 값을 전달하는 형태의 함수호출, 인자로 받은 값을 복사하여 처리한다.

장점 : 복사하여 처리하기 때문에 매개변수로 전달된 값이 보존이 되고 안전하다.

단점 : 복사를 하기에 메모리 사용량이 늘어난다.

|  |
| --- |
| #include <stdio.h>  void swap\_v(x, y) {  int temp = x;  x = y;  y = temp;  }  void swap\_r(int \*x, int \*y) {  int temp = \*x;  \*x = \*y;  \*y = temp;  }  int main() {  int a = 10, b = 20, c=10, d=20;  swap\_v(a, b);  printf("swap\_v : %d %d\n", a,b);  swap\_r(&c, &d);  printf("swap\_r : %d %d", c, d);  } |

텍스트이(가) 표시된 사진

자동 생성된 설명 

- main 함수의 argc, argv에 대해 조사 및 실습

프로그램 실행 시 메인 함수로 전달할 인자를 열거할 수 있으며, 메인 함수에서도 인자를 전달받을 수 있도록 제한된 형태의 매개변수 선언이 가능하다.

int argc (argument count) : 메인 함수에 전달되는 문자열의 개수(공백 기준)

char \*argv[] (argument vector) : 메인 함수에 전달되는 실질적 정보, 문자열을 담을 수 있는 포인터 배열 \*첫번째 문자열은 실행된 파일명

소스코드

|  |
| --- |
| #include <stdio.h>  int main(int argc, char\* argv[]) {  int i = 0;  printf("전달된 문자열의 수 : %d \n", argc);  for (i = 0; i < argc; i++) {  printf("%d번째 문자열 : %s \n", i + 1, argv[i]);  }  return 0;  } |

텍스트이(가) 표시된 사진

자동 생성된 설명

프로그램 실행과정에서 입력한 내용이 프로그램 안으로 전달되어 출력이 이루어졌음을 확인할 수 있다.

- 메모리 영역 조사

**코드 영역(Code Area)**

실행할 프로그램의 코드가 저장되는 메모리 공간

CPU는 코드 영역에 저장된 명령문들을 하나씩 가져가서 실행한다.

**데이터 영역(Data Area)**

**초기값이 있는 전역변수**와 static으로 선언되는 **static변수**가 할당된다. Code영역이 끝나는 곳에서 시작하며 이 영역에 할당되는 변수들은 프로그램의 시작과 동시에 메모리 공간에 할당되어 프로그램 종료 시까지 남아있다.

**Bss 영역**

**초기값이 없는 전역변수, static 변수**가 기본값 0을 가지고 할당된다. Data 영역이 끝나는 곳에서 Bss 영역이 시작된다.

**스택 영역(Stack Area)**

**지역변수**와 **매개변수**가 할당된다. 이 영역에 할당되는 변수들은 선언된 함수를 빠져나가면 소멸된다. 스택은 LIFO 방식으로 동작해 가장 마지막에 생성된 데이터를 가장 먼저 읽을 수 있다. 메모리의 높은 주소부터 시작하며, Heap 영역이 끝나는 곳에서 서로 만난다.

스택 프레임 : 스택 영역에 차례대로 저장되는 함수의 호출정보로 함수가 호출되면 스택에는 함수의 매개변수, 호출이 끝난 뒤 돌아간 반환 주소값, 함수에서 선언된 지역변수 등이 저장된다.

스택 오버플로우 : 스택 프레임에 모든 공간을 다 차지하고 더 이상의 여유 공간이 없을 경우 발생한다. C언어에서는 스택 오버플로우가 발생하면 강제 종료시킨다.

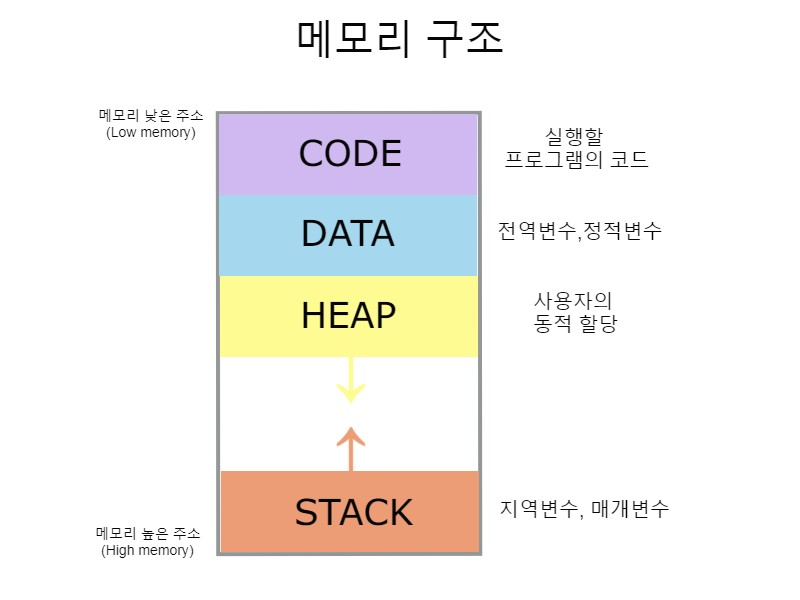
**힙 영역(Heap Area)**

필요에 의해 동적으로 메모리를 할당하고자 할 때 위치하는 메모리 영역, Data 영역이 끝나는 곳부터 시작되고 메모리의 낮은 주소에서 높은 주소 방향으로 할당된다. 메모리를 동적할당 하지 않아도 프로그램은 정상 작동하지만 메모리 관리의 효율성과 시스템 성능을 위해 사용된다. 동적할당이란 프로그램 수행도중 메모리의 크기를 필요한만큼 임의적으로 할당하는 것이다. 이 영역에 데이터를 저장하기 위해서는 malloc( ), free( ) 함수 등을 사용한다. int형 10개의 크기를 가진 배열만큼의 데이터를 힙 영역에 할당 받기 위해서는 다음과 같이 사용한다.

int\* a = (int \*)malloc(sizeof(int) \* 10);

프로그램이 종료되면 운영체제에 의해 할당된 메모리 공간 전체를 반환

--> 전역변수 소멸 시점



- static 변수 조사, 실습

static 지역 변수

지역변수에 static 선언이 붙게 되면, 전역변수의 성격을 지니는 변수가 된다.

static 자료형 변수이름;

선언된 함수 내에서만 접근이 가능하다. (지역변수의 특성)

프로그램 종료 시까지 메모리 공간에 존재한다. (전역변수의 특성)

static 전역 변수

전역 변수에 static을 붙이면 변수의 범위를 파일(지역) 범위로 제한하는 효과를 낸다.

static으로 선언한 변수는 따로 초기화를 하지 않아도 0으로 자동 초기화가 된다.

일반 지역 변수는 스택 영역에 저장되어 함수 호출시에 메모리에 할당되며 함수 종료시에 메모리에서 사라진다. 하지만 static 변수는 데이터 영역에 저장되어 프로그램 시작할 때 할당되며 프로그램 종료시에 사라진다.

소스코드

|  |
| --- |
| #include <stdio.h>  void count() {  static int a;  int b = 0;  a++;  b++;  printf("a : %d b : %d\n", a, b);  }  int main() {  count();  count();  count();  } |

출력 결과

텍스트이(가) 표시된 사진

자동 생성된 설명

- Calling Convention: cdecl 와 stdcall 에 대해 조사

함수 호출 규약 - 함수의 호출 방식에 대한 약속

Caller : 호출자, 함수를 호출한 대상

Callee : 피호출자, 호출을 당한 함수

호출자는 피호출자에게 인자를 전달해줘야 하고, 피호출자의 실행이 종료되면 반환 값을 전달받아야 한다.

구분방법

-인자 전달 순서 ( 왼쪽 -> 오른쪽 or 오른쪽 -> 왼쪽)

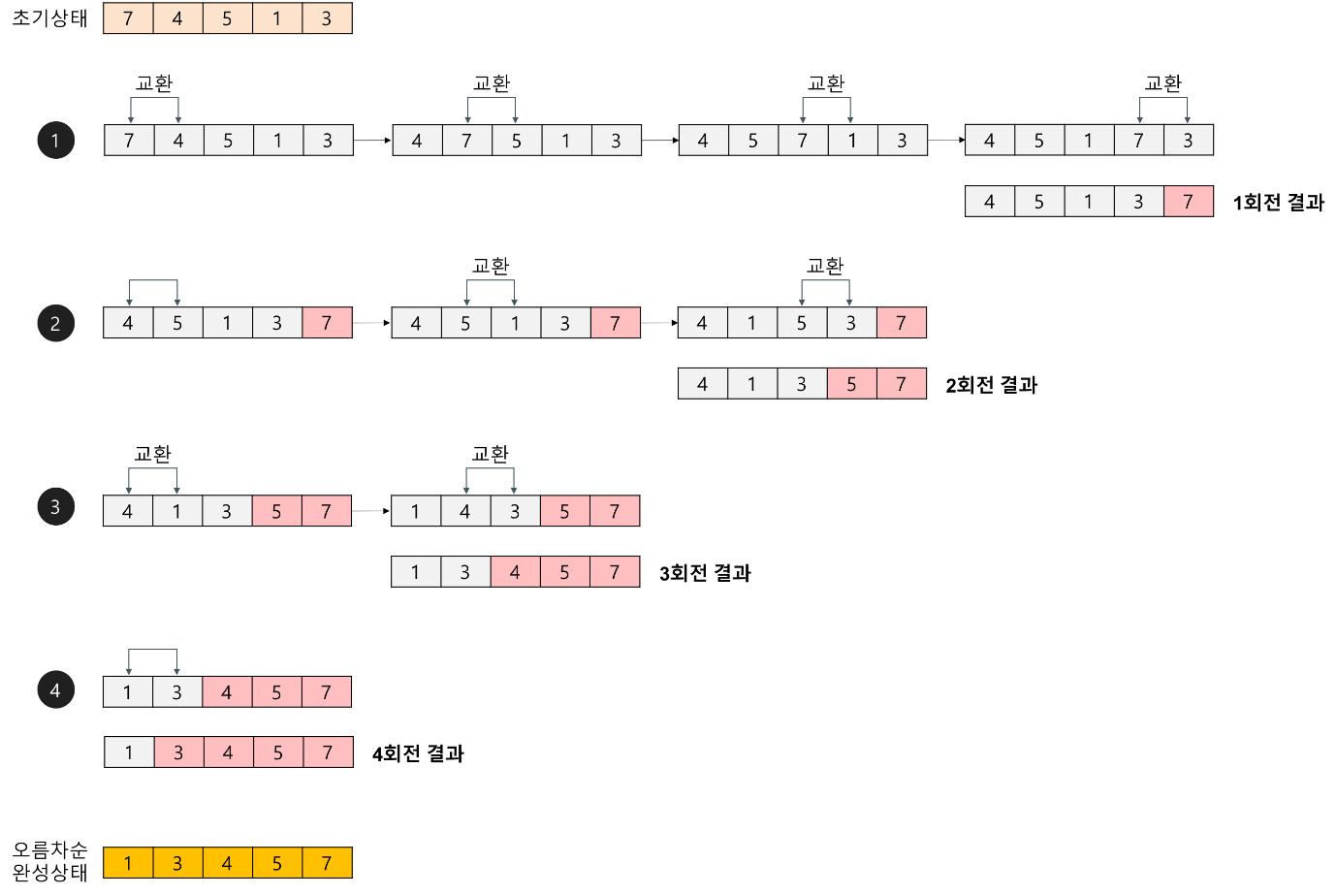
-인자 전달 방법 (스택 or 레지스터)

-스택 프레임을 정리하는 방법 (Caller or Callee)

|  |  |
| --- | --- |
| **cdecl** | **stdcall** |
| -인자 전달 순서 : 오른쪽 -> 왼쪽  -인자 전달 방법 : 스택  -스택 프레임을 정리하는 방법 : Caller  -가변 개수의 인자 전달 가능 | -인자 전달 순서 : 오른쪽 -> 왼쪽  -인자 전달 방법 : 스택  -스택 프레임을 정리하는 방법 : Callee  -가변 개수의 인자 전달 불가  호출된 함수가 스택에서 제거해야 할 인자의 개수를 정확히 알아야 하기 때문 |

- 버블 정렬 조사

서로 이웃한 두 원소를 비교하여 가장 큰 데이터를 가장 뒤로 보내 정렬하는 방식, 인접한 두 원소가 크기 순서대로 되어 있지 않으면 서로 교환한다.



- 재귀 함수 조사

함수 내에서 자기 자신을 다시 호출하는 함수

똑같은 구조의 함수를 반복해서 사용해야할 때 이용한다. 재귀함수를 적용하면 연산식은 재귀함수를 이용하여 연산을 끝없이 반복해 무한 루프 상태에 빠질 수 있기에 연산을 끝낼 수 있는 탈출 조건을 정해주어야 한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

32bit vs 64bit 차이 조사

근본적으로 CPU의 차이이다. 32bit, 64bit는 레지스터의 크기를 의미하는 것이고 bit 수가 클수록 한번에 더 많은 정보를 저장할 수 있으므로 처리 속도도 빨라진다.

2^32=4294967296

2^64=18446744073709551616

가장 큰 차이점은 메모리 인식률로 32비트는 4GB까지 가능하지만 64비트는 4GB이상의 RAM을 인식할 수 있다. 따라서 32비트는 10GB RAM을 꽂아도 4GB만 표시된다.

호환성

64비트로 만들어진 게임을 32비트 윈도우에 설치한다면 64비트만큼 필요한데 32비트만큼만 있기에 실행 오류가 발생한다. 반대로 64비트 윈도우에 32비트 게임을 실행한다면 64비트만큼의 여유가 있어 32비트만큼의 게임파일을 읽고 메모리에 기록 후 시작해도 필요한 모든 데이터가 있기에 정상적으로 실행된다.

-상대경로, 절대경로 조사

경로 : 파일 시스템 내에서 특정 파일의 위치, 이름을 나타내는 양식

상대경로 : 현재 위치를 기준으로 하여 목적지까지의 상대적인 경로

최상위 /를 거치지 않고도 이동이 가능

절대경로 : 처음부터 시작하여 경유한 경로를 전부 기입하는 방식의 절대적인 경로

최상위 /를 포함

정적인 문자열로 특정 파일 위치를 정확히 알려준다.

절대경로 상대경로 차이점

|  |  |  |
| --- | --- | --- |
|  | 절대경로 | 상대경로 |
| 컴파일 속도 | 느리다 | 빠르다 |
| 해당 소스의 위치 변환 시 | 경로 다시 지정 | 기준이 된 폴더의 구성이 달라지지 않는 한 경로지정을 할 필요가 없다. |
| 분실 가능성 | 낮다 | 높다 |
| 사용 | 외부 파일 참조시 | 내부 파일 연결시 |
| 예시 | /home/usr/ubuntu/workspace/ mypage/env/static/background.jpg | ../static/background.jpg |

/ : 루트

./ : 현재 위치

../ : 현재 위치의 상단 폴더

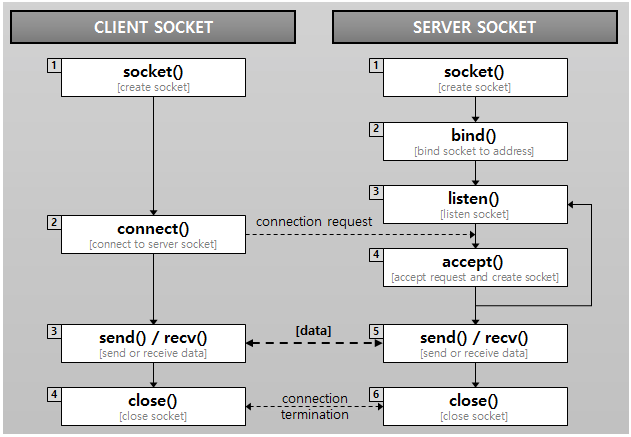
- Socket 개념, Socket 통신 과정 조사

Socket은 통신을 위한 일종의 통로라고 생각할 수 있다. 소켓은 프로세스가 드넓은 네트워크 세계로 데이터를 내보내거나 혹은 그 세계로부터 데이터를 받기 위한 실제적인 창구 역할을 한다. 프로세스가 데이터를 보내거나 받기 위해서는 반드시 소켓을 열어서 소켓에 데이터를 써보내거나 소켓으로부터 데이터를 읽어들여야 한다. 소켓은 프로토콜, IP주소, 포트 번호로 정의된다.

Client socket & Server socket

두 개의 시스템(또는 프로세스)이 소켓을 통해 네트워크 connection을 맺기 위해서는 최초 어느 한 곳에서 그 대상이 되는 곳으로 연결을 요청해야 한다.

이러한 connection을 위해서는 IP주소와 포트 번호를 알아야한다.



클라이언트 소켓 처리 과정

* 처음 소켓을 생성(socket())한 다음, 서버 측에 연결을 요청(connect())한다. 통신할 서버의 설정된 IP와 포트번호에 통신을 시도
* 서버 소켓에서 연결이 받아들여지면 데이터를 송수신(send(), recv())하고, 모든 처리가 완료되면 소켓을 닫는다.(close())

서버 소켓 처리 과정

* 처음 소켓을 생성(socket())한 다음, 서버가 사용할 IP주소와 포트 번호를 생성한 소켓에 결합(bind())시킨다.
* 클라이언트로부터 연결 요청이 수신되는지 주시(listen())하고, 요청이 수신되면 요청을 받아들여(accept()) 데이터 통신을 위한 소켓을 생성한다.
* 새로운 소켓을 통해 연결이 수립되면, 클라이언트와 마찬가지로 데이터를 송수신할 수 있다. (send(), recv())
* 데이터 송수신이 완료되면 소켓을 닫는다.(close())

소켓 종류

연결형(TCP)인지 비연결형(UDP)인지에 따라 소켓 통신 과정이 다르다.

연결 지향 프로토콜(TCP)

양방향으로 바이트 스트림을 전송하며 연결 지향성이다. 오류 수정, 전송 처리, 흐름 제어를 보장하고 송신된 순서에 따라 중복되지 않게 데이터를 수신해서 오버헤드가 발생한다. 그래서 소량의 데이터보다 대량의 데이터 전송에 적합하다. 데이터의 신뢰도가 중요하다고 판단될 때 주로 사용한다.

비연결형 지향 프로토콜(UDP)

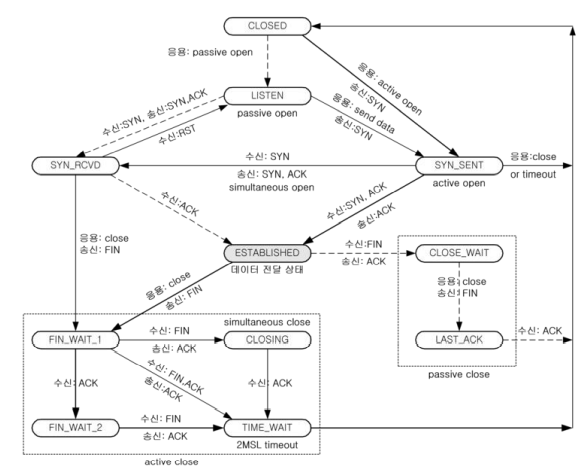
비연결형으로 데이터 크기에 제한이 있고 확실하게 전달이 보장되지 않는다. 발신자가 대상 수신자 주소를 향해 데이터를 일방적으로 전송하는 구조다. 데이터가 손실되어도 오류가 발생하지 않는다. 실시간 멀티미디어 정보를 처리하기 위해 주로 사용된다.

Socket 통신 특징

HTTP통신과 Socket통신의 차이 : Socket 통신은 클라이언트와 서버가 특정 포트를 통해 실시간 양방향 통신을 하는 방식이다. HTTP는 클라이언트가 요청을 보내는 경우에만 서버가 응답하는 단방향 통신이다.

서버와 클라이언트가 실시간으로 데이터를 주소 받는 상황이 필요한 경우에 사용한다.

-통신(TCP 프로토콜 통신) 과정



- 네트워크 포트 개념

포트 : 논리적인 접속 장소

인터넷 프로토콜인 TCP/IP를 사용할 때 클라이언트가 네트워크 상의 특정 서버 프로그램을 지정하여 사용한다.

IP : 컴퓨터를 찾을 때 필요한 주소

Port : 컴퓨터 안에서 프로그램을 찾기 위한 수단

사용 목적에 따라 지정된 논리적 주소인 포트 번호로 연결된 두 개의 컴퓨터 사이에서 네트워크를 이용한 통신 시, 발신 컴퓨터에서 출발한 패킷은 TCP/IP 각 계층을 거치면서 최종 목적지 주소를 가지고 있는 컴퓨터에 도착하고 패킷을 수신한 컴퓨터는 패킷 안에 있는 데이터를 응용 프로그램에 전달한다. 응용 프로그램에 이미 정해져 있는 포트 번호를 이용하여 전송 계층에서 응용 프로그램을 구분한다.

PORT 0 ~ 1023 Well-known port

특수용도로 지정되어 있기 때문에 가급적 개인적 테스트용 프로그램을 개발 시에는 0~1023사이의 포트번호는 피하는 것이 좋다.

|  |
| --- |
| 21번: FTP  22번: SSH  23번: TELNET  -> 인터넷을 통하여 원격지의 호스트 컴퓨터에 접속할 때 지원되는 인터넷 표준 프로토콜  25번: SMTP[Simple Mail Transfer Protocol]  -> 인터넷에서 전자우편을 보낼 때 이용하게 되는 표준 통신 규약  53번: DNS  61번: SNMP[Simple Network Management Protocol]  -> 네트워크 장비를 관리 감시하기 위한 목적으로 TCP/IP 상에 정의된 응용 계층 표준 프로토콜  80번: HTTP [hypertext transfer protocol]  -> 인터넷에서, 웹 서버와 사용자의 인터넷 브라우저 사이에 문서를 전송하기 위해 사용되는 통신 규약  110번: POP3 [Post Office Protocol version 3]  -> 인터넷에서 전자 우편을 가져오기 위한 프로토콜  115번: SFTP  135번: RPC  139번: NetBIOS  143번: IMAP [internet messaging access protocol]  -> 인터넷 메일 서버에서, 메일을 읽기 위한 인터넷 표준 통신 규약의 한가지. POP3보다도 유연하고 뛰어난 성능  194: IRC  443: HTTPS(SSL) [Hypertext Transfer Protocol over Secure Socket Layer]  -> 월드 와이드 웹 통신 프로토콜인 HTTP의 보안이 강화된 버전  445: SMB  3389: 원격데스크탑 연결 |

PORT 1024 ~ 49151 registered port

PORT 49152 ~ 65535 dynamic port

- 소켓통신 코드 분석

서버 소스코드

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <unistd.h>  #include <arpa/inet.h>  #include <sys/types.h>  #include <sys/socket.h>  #define BUFF\_SIZE 1024  int main(){  int server\_socket;  int client\_socket;  int client\_addr\_size;  struct sockaddr\_in server\_addr; //sockaddr\_in : 소켓 주소의 틀을 형성해주는 구조체  struct sockaddr\_in client\_addr; //accept()에서 사용  char buff\_rcv[BUFF\_SIZE+5];  char buff\_snd[BUFF\_SIZE+5];  server\_socket = socket(PF\_INET,SOCK\_STREAM,0); //ipv4 도메인,TCP연결지향형 소켓 생성  if(server\_socket==-1){ // 0이상의 값 : 소켓 디스크립터  printf("server socket fail\n");  exit(1);  }  else{printf("소켓 생성 완료\n");}  memset(&server\_addr,0,sizeof(server\_addr)); //주소 초기화  server\_addr.sin\_family = AF\_INET;//ipv4  server\_addr.sin\_port=htons(49445);//port  server\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);//ip주소  //소켓, 서버 주소 바인딩 성공시 0 리턴  if(bind(server\_socket,(struct sockaddr\*) &server\_addr,sizeof(server\_addr))==-1){  printf("bind() error\n");  exit(1);  }  else{printf("바인딩 완료\n");}  //연결 대기열 5개 생성 성공시 0 리턴  if(listen(server\_socket,5)==-1){  printf("listen() fail\n");  exit(1);  }  //else{printf("대기열 생성 완료\n");}  while(1){//클라이언트로부터 요청이 오면 연결 수락  client\_addr\_size = sizeof(client\_addr);  client\_socket = accept(server\_socket,(struct sockaddr\*)&client\_addr,&client\_addr\_size);  if(client\_socket==-1){  printf("accept() fail\n");  exit(1);  }  //else{printf("연결수락\n");}  read (client\_socket,buff\_rcv,BUFF\_SIZE);//클라이언트로부터 전송된 자료 읽기  printf("receive: %s\n",buff\_rcv);  printf("데이터전송\n");  sprintf(buff\_snd,"%d : %s",strlen(buff\_rcv),buff\_rcv);//수신된 데이터의 길이를 구해 전송 데이터 준비  write(client\_socket,buff\_snd,strlen(buff\_snd)+1); //클라이언트로 자료 송신(NULL 포함)  close(client\_socket); //통신 종료  }  } |

클라이언트 소스코드

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <unistd.h>  #include <arpa/inet.h>  #include <sys/types.h>  #include <sys/socket.h>  #define BUFF\_SIZE 1024  int main(int argc,char \*\*argv){  int client\_socket;  struct sockaddr\_in server\_addr;  char buff[BUFF\_SIZE+5];  client\_socket = socket(PF\_INET,SOCK\_STREAM,0); //ipv4 도메인,TCP연결지향형 소켓 생성  if(client\_socket==-1){  printf("socket create fail\n");  exit(1);  }  memset(&server\_addr,0,sizeof(server\_addr)); //인자로 받은 서버 주소 정보 저장  server\_addr.sin\_family = AF\_INET;//서버주소체계 ipv4  server\_addr.sin\_port = htons(49445);//포트번호  server\_addr.sin\_addr.s\_addr=inet\_addr("127.0.0.1");//서버주소IP 저장  //클라이언트 소켓부분에 서버 연결  if(connect(client\_socket,(struct sockaddr\*)&server\_addr,sizeof(server\_addr))==-1){  printf("connect:fail\n");  exit(1);  }  write(client\_socket,argv[1],strlen(argv[1])+1);//접속 성공시 데이터 전송(NULL 포함)  read(client\_socket,buff,BUFF\_SIZE); //자료 수신 후 출력  printf("%s\n",buff);  close(client\_socket);  return 0;  } |

클라이언트, 서버 통신 실행

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

친구 관리 프로그램

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명