2022 CAT-CERT 스터디 보고서

컴퓨터정보공학부 202221168 김지혜

2022.03.30 #C\_study-1

+++실습

소스코드

#include <stdio.h>

int main() {

printf("\t안녕하세요?\n C언어 스터디 \\첫 번째\\ 시간입니다.\n");

return 0;

}

출력결과

텍스트이(가) 표시된 사진

자동 생성된 설명

IF문 실습1

소스코드

#include <stdio.h>

int main()

{

int a, b, opt;

int res;

printf("1.덧셈 2.뺄셈 3.곱셈 4.나눗셈 \n");

printf("선택 : ");

scanf("%d", &opt);

printf("두 개의 정수 입력 : ");

scanf("%d %d", &a, &b);

if (opt == 1)

res = a + b;

else if (opt == 2)

res = a - b;

else if (opt == 3)

res = a \* b;

else if (opt == 4)

res = a / b;

printf("결과 : %d \n", res);

return 0;

}

출력 결과

텍스트이(가) 표시된 사진

자동 생성된 설명

IF문 실습2

소스코드

#include <stdio.h>

int main()

{

int a;

printf("숫자 입력 : ");

scanf("%d", &a);

if (a % 15 == 0)

printf("%d는 3과 5의 배수입니다.\n", a);

else

printf("%d는 3과 5의 배수가 아닙니다.\n", a);

return 0;

}

출력결과

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

IF문 실습3

소스코드

#include <stdio.h>

int main()

{

int year;

printf("연도 입력 : ");

scanf("%d", &year);

if (year % 4 == 0 && year % 100 != 0)

printf("%d년은 윤년입니다.\n", year);

else if (year % 400 == 0)

printf("%d년은 윤년입니다.\n", year);

else

printf("%d년은 윤년이 아닙니다\n", year);

return 0;

}

출력결과

텍스트이(가) 표시된 사진

자동 생성된 설명

switch문 실습

소스코드

#include <stdio.h>

int main()

{

int month;

printf("Month Input : ");

scanf("%d", &month);

switch (month) {

case 1: case 3: case 5: case 7: case 8: case 10: case 12:

printf("31일까지 있습니다.\n");

break;

case 2:

printf("28일까지 있습니다.\n");

break;

case 4: case 6: case 9: case 11:

printf("30일까지 있습니다.\n");

break;

}

}

출력 결과

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

삼항연산자 실습

소스코드

#include <stdio.h>

int main()

{

int a, b;

scanf("%d %d", &a, &b);

printf("%d\n", (a > b) ? b : a);

}

출력 결과

텍스트이(가) 표시된 사진

자동 생성된 설명

실습 과제 1

소스코드

#include <stdio.h>

int main()

{

int n;

scanf("%d", &n);

printf("%d\n", (n \* (n + 1) / 2));

return 0;

}

출력 결과

텍스트이(가) 표시된 사진

자동 생성된 설명

실습 과제2

소스코드

#include <stdio.h>

int main()

{

int a, b, c, d;

scanf("%d %d", &a, &b);

scanf("%d %d", &c, &d);

if ((a + d) >= (b + c))

printf("%d\n", (b + c));

else if ((a + d) <= (b + c))

printf("%d\n", (a + d));

}

출력 결과

텍스트이(가) 표시된 사진

자동 생성된 설명

실습 과제 3

소스코드

#include <stdio.h>

int main()

{

int a, b, c;

scanf("%d, %d, %d", &a, &b, &c);

if (a < b) {

if (b < c) {

printf("%d\n", b);

}

else if ((b > c) && (c > a)) {

printf("%d\n", c);

}

}

else if (a > b) {

if ((a > c) && (c > b)) {

printf("%d\n", c);

}

else if (a < c) {

printf("%d\n", a);

}

}

}

출력 결과

텍스트이(가) 표시된 사진

자동 생성된 설명

실습 과제 4

소스코드

int main()

{

char one, two, three;

scanf("%c %c %c", &one, &two, &three);

if (one < 97)

one += 32;

if (two < 97)

two += 32;

if (three < 97)

three += 32;

if ((one < two) && (two < three))

printf("O\n");

else

printf("X\n");

return 0;

}

출력결과

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

조사 과제

#1 컴파일러에 대해 조사하기

-**컴파일러** : 특정 프로그래밍 언어(고급 언어)로 쓰여 있는 문서를 다른 프로그래밍 언어나 기계어(컴퓨터 언어)로 옮기는 번역 프로그램, 언어에 따라 다양한 컴파일러 종류가 존재함

-

-**인터프리터** : 고급언어로 작성된 프로그램을 한 줄 단위로 받아들여 번역, 번역과 동시에 프로그램을 한 줄 단위로 실행

CPU의 사용시간의 낭비가 큼

컴파일러는 고급언어를 한번에 번역해서 번역 시간이 오래걸림

한번 번역하면 실행파일이 생성되어 다음에 실행할 때 기존에 생성된 파일 실행 -> 이때는 인터프리터에 비해 실행이 빠름, 목적 프로그램을 생성함

인터프리터는 한줄씩 번역해서 번역시간은 빠름, 실행할 때마다 같은 번역을 진행해야하기 때문에 컴파일러에 비해 느림, 목적 프로그램을 생성하지 않음

\*목적 프로그램 : 원시 언어에서 목적 언어로 번역된 프로그램

\*소스코드- 원래의 문서

\*목적코드- 출력된 문서

#2 디버그모드와 릴리즈모드의 차이 조사

\*디버깅 : 시스템의 논리적 오류나 버그를 찾아 내고 제거하는 것

-**디버그 모드** : 사용자가 좀 더 쉽게 디버깅 할 수 있도록 도와주는 기능

같은 문자열 상수라면 같은 메모리 영역을 그대로 사용하고, 실행파일에 디버깅 정보를 삽입함. 필요한 정보를 실행할 때 계속 체크 해서 속도가 느림

디버깅 정보가 있기에 실행파일 상태 확인 가능

-**릴리즈 모드** : 프로그램을 배포하기위해 컴파일 하는 모드, 디버깅 정보 없이 순수한 소스코드 자체의 기능만 컴파일되어 실행파일로 만들어짐

같은 문자열 상수여도 다른 메모리 공간에 할당함. 디버깅 정보를 삽입하지 않아 실행파일의 크기를 줄여주고, 코드를 최적화 시킴. 속도나 크기가 유리하기 때문에 메모리 점유율도 낮고 실행도 빠름

-두 모드에서 메모리를 할당하면 요청 크기만큼 받는 메모리의 초기값이 다름

#3 리틀엔디안 vs 빅엔디안 조사

**엔디안** : 1차원 공간(컴퓨터의 메모리)에 여러 개의 연속된 대상을 배열하는 방법

**리틀엔디안** : 최하위 바이트부터 차례대로 저장 – 작은 단위의 바이트가 앞에 나오는 방법

* 메모리에 저장된 값의 하위 바이트들만 사용할 때 별도의 계산이 필요가 없음.
* Intel 계열의 프로세서에 사용됨

**빅 엔디안** : 최상위 바이트부터 차례대로 저장 - 큰 단위의 바이트가 앞에 나오는 방법

* 소프트웨어의 디버그를 편하게 해줌. 디버깅 과정에서 메모리의 값을 보기 편하게 함
* Unix의 RISC 프로세서에서 사용하고, 네트워크에서도 사용함

-**빅 엔디안**은 사람이 숫자를 읽고 쓰는 방법과 동일하고, 수치를 앞에서 차곡차곡 스택에 집어 넣음. **리틀 엔디안**은 숫자 뒤에서부터 넣기 때문에 속도가 느림

-두 숫자를 계산할 경우 낮은 자리수에 있는 계산을 보고 올림수가 있는지 없는지 판단하고 다음 자리수를 계산하기 때문에 수치 계산은 리틀 엔디안이 더 빠르다

빅엔디안일 때 : 0x41,0x42,0x43,0x44 0x41424344

리틀엔디안 : 0x44,0x43,0x42,0x41 0x44434241

#4 .C파일이 실행파일이 되기까지의 과정 조사

1. 코드 작성
2. 전처리

전처리기가 소스코드를 컴파일하기 좋게 다듬는 과정, 소스코드에서 “#”으로 시작한는 지사자를 처리함

* #include는 header file(헤더파일) 내용을 읽어서 필요한 내용을 지시자가 있는 위치에 붙여놓음, #define은 같은 내용을 만나면 해당내용으로 치환(매크로 치환 작업)

1. 컴파일

전처리가 끝난 소스코드를 기반으로 어셈블리어(기계어에 가까운 언어)로 번역하는 단계

1. 어셈블러

번역된 어셈블리어를 기계어로 최종적으로 번역하는 단계

어셈블러 소스를 바탕으로 목적 파일을 생성함-사람이 볼 수 있는 형태가 아닌 기계어 코드가 됨

\*목적 파일로만은 라이브러리 파일에 있는 함수를 사용할 수 없다

1. 링크

여러 개로 흩어져 있는 목적파일을 합쳐서 하나의 실행파일을 만드는 것

* 목적 파일은 직접 실행할 수 있는 파일이 아니기 때문에 실행파일로 만들기 위해서 하는 작업

\*C언어는 분할 컴파일이 가능함!

* 또한 C언어의 표준 라이브러리들도 여러 개의 목적 파일을 모아서 하나의 파일로 만들어 둠

1. 실행파일 생성

#5 기계어와 어셈블리어에 대해 조사

**기계어** : 컴퓨터가 사용하는 언어 즉, 컴퓨터가 읽을 수 있는 2진 숫자로 이루어진 언어

**어셈블리어** : 기계어와 일대일 대응이 되는 컴퓨터 프로그래밍의저급 언어(기계어와 가까움)

-어셈블리 명령어들이 프로세서의 명령어들과 하드웨어 구조에 맞추어 설계되었기 때문에 각 프로세서에 특화 되어있음

* Windows -> Intel 문법
* Linux -> AT&T 문법

-특정 프로세서에만 있는 레지스터에 바로 접근가능

**\*레지스터** : CPU의 요청을 처리하는 데이터의 임시공간

CPU에 직접 연결되어 있어서 연산속도가 빠름

-코드 하나하나의 동작을 모두 제어하고 생각한대로 할 수 있음

-일반적인 컴파일러가 제공하는 환경과 생성하는 코드의 규칙을 무시한 채 작업할 수 있음 – 속도의 최적화를 위해

-버그 발생 확률이 높고 추적해서 잡기 힘듦

-코드 이해와, 수정이 어려움