**Quiz #1 (프로그래밍 언어론) (기한 5/22)**

학번: 20204101 이름: 신동욱

**1. 다음에 답하시오. (1\*10 = 10)**

**a. C언어의 스트링 길이 선택 사항은 무엇인가? 실행시간 서술자가 필요한가? 그 이유를 주라.**

C언어는 제한된 동적 길이 스트링이다. C언어의 문자열의 끝은 널 문자(\0)로 되어 있기 때문에 실행 시간 서술자가 필요 없다.

**b. Java에서 스트링을 지원하는 2개의 클래스는 무엇인가?**

String, StringBuffer 클래스로 지원한다

**c. 레코드와 튜플간의 주요 차이점은 무엇인가?**

레코드는 각 원소들이 명명되나, 튜플은 각 원소들이 명명되지 않는다.

**d. Java의 참조 타입이 C의 포인터에 비해서 갖는 장점은 무엇인가?**

안정성이 높다. Java의 참조 타입은 허상 참조가 발생하지 않는다.

**e. C++에서 형식 매개변수로 포인터보다 참조 변수를 사용하는 것이 왜 더 나은가?**

참조 변수를 사용하는 것이 판독성, 안전성이 향상된다.

**f. Java의 참조에 대한 산술 연산이 의미가 없는 이유는 무엇인가?**

참조 변수는 메모리의 객체나 값을 참조하기 때문에 참조에 대한 산술 연산은 무의미하다.

**g. Java가 강타입 언어가 아닌 이유는 무엇인가?**

거의 강타입 언어이나, 사용자에 의한 명시적 타입 캐스팅에 의한 오류 가능성이 있다.

**h. C의 공용체가 자유 공용체라 불리는 이유는 무엇인가?**

타입 검사를 지원하지 않기 때문이다.

**i. Java에서 허상 참조가 발생할 수 있는가? 그 이유는 무엇인가?**

자바는 프로그램이 종료 될 때 Garbage Collector가 참조를 알아서 회수하기 때문에 허상 참조가 일어나지 않는다

**j. C가 구조 타입 동등을 적용하는 타입은 무엇인가?**

struct, union, enum이 다른 파일에 정의되어 있으면 구조 타입 동등을 적용한다

**a. 다음 배열의 4가지 유형을 정의하라: 정적, 고정 스택-동적, 고정 힙-동적, 힙-동적 배열. 또한, 각각의 장점은 무엇인가?**

정적 배열 : static int arr[5]

장점 : 효율적이다. 실행 중 배열 공간 유지됨

고정 스택-동적 배열 : int arr[5]

장점 : 기억공간이 효율적이나, 할당/반환에 대해 부담

고정 힙-동적 배열 : int \*p = (int\*) malloc(sizeof(int) \* 5)

장점 : 첨자 범위와 기억 공간 바인딩이 실행 중 요청될 때 이루어짐

힙-동적 배열 : ArrayList<Integer> arr = new List<>();

장점 : 유연성이 높다, 하지만 할당/회수에 부담이 크다

**b. 다차원 배열은 행-우선 순서로 저장되거나 열-우선 순서로 저장될 수 있다. 각 경우에 대해서 a[i,j]에 대한 접근 함수를 작성하라.**

행 우선 순서 : (a[i][j]) = a + ( i \* 열의 크기 + j) \* sizeof(타입)  
열 우선 순서 : (a[i][j]) = a + ( j \* 행의 크기 + i) \* sizeof(타입)

**c. Java의 배열 서술자에서 필요한 항목들은 무엇인가? 그리고 이들이 언제 저장되어야 하는가? (컴파일 시간 또는 실행시간)**

배열, 원소 타입, 인덱스 타입, 인덱스 하한, 인덱스 상한, 주소  
자바는 컴파일 시간에 바인딩 된다.

**d. C++와 Java의 참조 타입 변수를 비교하라.**

C++ : 항상 묵시적으로 역 참조되는 상수 포인터, 함수의 형식 매개변수에서 사용  
Java : 포인터 연산을 제한한 참조 타입, 객체에 대한 참조

**e. C, C++, Java를 강타입 언어 관점에서 비교하라.**

C, C++ : 강타입이 아니다. 공용체는 컴파일 시 타입 검사를 하지 않는다.  
Java : 거의 강타입이나 사용자에 의한 명시적 타입 캐스팅에 의한 오류 가능성이 있다.

**3. 다음에 답하시오 (5\*2 = 10)**

**a. C, C++, Java의 열거 타입을 비교하라. 비교시에 각 언어의 코드 예제를 들면서 설명하라. (코드를 실행시켜서 확인하라) (비교 사항은 아래에 작성하고, 코드와 실행 결과 화면을 첨부할 것)**

C : enum 키워드로 정의, 타입 안정성이 비교적 낮다. 열거형 변수를 출력하면 정수가 출력된다.

C++ : enum class 키워드로 정의, 타입 안정성을 제공한다. 열거형 변수를 출력하면 정수가 출력된다.

Java : enum 키워드로 정의, 타입 안정성을 제공한다. 열거형 변수를 출력하면 해당하는 문자열이 출력된다.

타입 안정성이란 변수나 값이 명시된 타입에 적법하게 사용되고, 잘못된 타입 변환으로 발생할 수 있는 오류를 방지하는 특성을 말한다.

또한 각 열거형 변수를 출력하는 부분에서 차이점이 존재한다. (제가 생각하는 가장 직관적인 큰 차이점 이라고 생각합니다)

**b. 행렬을 매개변수로 전달받아서 행렬 곱셈을 수행하여 그 결과를 반환하는 C 함수 matmul()을 다음 2 가지 방법으로 작성하라: 1) 배열 참조에 첨자만을 사용한다. 2) 배열 참조에 포인터와 포인터 산술 연산 만을 사용한다. 두 함수의 실행 시간을 측정하여 비교하라. 또한, 두 함수를 신뢰성의 관점에서 비교하라. 각 비교에 대해서 그 이유를 주라. (비교 사항은 아래에 작성하고, 코드와 실행 결과 화면을 첨부할 것)**

실행 시간 비교: 첨자를 통한 참조가 포인터 산술 연산을 통한 참조보다 더 빠르다. 포인터를 직접 산술 연산하는 것은 컴파일러가 정확하게 알 수 없다. 포인터를 직접 연산하는 것은 복잡한 계산을 포함한다. 이는 배열의 차원이 커질수록 더욱 복잡해진다.   
일반적으로 첨자를 통한 방식으로 배열에 접근하는 것을 사용자들은 선호했으며, 이에 맞춰 컴파일러의 최적화는 배열을 통한 방식에 대해 좀 더 최적화가 되어 있다.

신뢰성: 첨자를 통한 방식이 포인터 산술 연산 방식보다 가독성이 뛰어나다. 이는 배열의 차원이 높아질수록 두드러진다. 가독성은 즉 단순성과 연관되어 있기 때문에 첨자 방식이 신뢰성이 더 높다. 또한 포인터 산술 연산 방식은 잘못된 주소에 대한 접근이 발생할 확률이 더욱 높다.

C언어 열거형

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명enum Color {

RED, GREED, BLUE

};

int main()

{

Color c = RED;

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

}

C++ 열거형

enum class Color {

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명 RED, GREED, BLUE

};

int main()

{

Color c = Color::RED;

//cout << c; 컴파일 에러

cout << "Color: " << (int)c << endl;

}

Java 열거형

텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명class test {  
   
 enum Color {  
 *RED*, *GREED*, *BLUE* }  
   
 public static void main(String[] args) {  
 Color c = Color.*RED*;  
 System.*out*.println(c);  
 }  
}

#define N 3

void matmul\_1(int A[N][N], int B[N][N]) {

int C[N][N];

int i, j, k;

for (i = 0; i < N; i++) {

for (j = 0; j < N; j++) {

C[i][j] = 0;

}

}

for (i = 0; i < N; i++) {

for (j = 0; j < N; j++) {

for (k = 0; k < N; k++) {

C[i][j] += A[i][k] \* B[k][j];

}

}

}

}

void matmul\_2(int A[N][N], int B[N][N]) {

int C[N][N];

int i, j, k;

for (i = 0; i < N; i++) {

for (j = 0; j < N; j++) {

\*(\*(C + i) + j) = 0;

}

}

for (i = 0; i < N; i++) {

for (j = 0; j < N; j++) {

for (k = 0; k < N; k++) {

\*(\*(C + i) + j) += \*(\*(A + i) + k) \* \*(\*(B + k) + j);

}

}

}

}

int main() {

double start, end;

int A[N][N] = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };

int B[N][N] = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };

start = (double)clock() / CLOCKS\_PER\_SEC;

for(int i = 0; i < 1000000; i++)

matmul\_1(A, B);

end = (((double)clock()) / CLOCKS\_PER\_SEC);

printf("matmul\_1 시간 : %lf\n", (end - start));

start = (double)clock() / CLOCKS\_PER\_SEC;

for (int i = 0; i < 1000000; i++)

matmul\_2(A, B);

end = (((double)clock()) / CLOCKS\_PER\_SEC);

printf("matmul\_2 시간 : %lf\n", (end - start));

}

**실행 결과**





텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

각 함수 호출을 1,000,000번 씩 반복하여 호출하여 시간을 계산함.

첨자를 통한 참조가 포인터 산술 참조보다 더 빠른 것을 볼 수 있다.