**프로그래밍언어(가)**

**과제1-1**

**이름 : 허예림**

**학번 : 20182667**

**교수님 : 유재우 교수님**

**1. 연습문제 3,4,8,18번에 대해 각각 간단히 5-10줄 사이로 답하시오.**

**3. What arguments can you make for the idea of a single language for all programming domains?**

나는 모든 유형의 프로그래밍에 대해 일반화된 단일 언어를 사용하는 것은 좋다고 생각한다. 각 사례마다 다른 언어를 이해하는데 더이상 어려움이 없기 때문에 개발자와 사용자 모두 언어에 대해 더욱 깊이 이해할 수 있게 될 것이다. 또한 동일한 언어로 작성된 프로그램을 서로 호환 가능하게 만들 수 있으므로 다른 플랫폼에 대한 포트가 필요하지 않게 될 것이다. 그리고 프로그래밍 교육 및 컴파일러 구매 및 유지관리 비용을 대폭 절감할 수 있을 것이다. 이는 프로그래머 모집을 쉽게 할 수 있게 되며 수많은 언어를 의존하는 소프트웨어 개발 보조 도구의 개발을 정당화 할 것이다.

**4. What arguments can you make against the idea of a single language for all programming domains?**

나는 모든 프로그래밍 영역에 단일 언어를 사용하면 각 개발자 및 사용자의 이해를 도울 수 있다고 생각한다. 그렇기에 단일 언어를 사용하면 시간을 절약하고 효율적으로 사용할 수 있게 될 것이다. 그러나 프로그래밍 영역에서 단일 언어를 사용하는 것은 문제가 될 수 있다. 프로그래밍을 위해 다양한 언어가 존재하는 장점은 다른 개발자가 백그라운드에서 발생하는 알고리즘과 프로세스를 알지 못하기 때문에 개발자가 자신의 아이디어를 훨씬 더 안전하게 표현할 수 있다는 점이 있다. 따라서, 다양한 종류의 프로그래밍 언어가 존재하는 것이 서로의 개발이 더 나아지도록 도와줄 것이다.

**8. Many languages distinguish between uppercase and lowercase letters in user-defined names. What are the pros and cons of this design decision?**

이 설계 결정의 장점은 프로그램이 코드를 더 정확하게 읽을 수 있게 해주며 프로그램 코드를 보다 쉽게 볼 수 있게 해준다. 또한 언어를 보다 전문적으로 보이게 해준다. 즉, 대소문자를 구분하는 식별자를 사용하면 더 다양한 식별자를 가능하게 해주고 프로그램에서 다르게 보이는 두 개의 이름이 컴파일러에 의해 동일하게 사용될 때 프로그래머에게 혼동을 주는 것을 막을 수 있다. 한편, 단점으로는 대문자 또는 소문자를 잘못 쓰면 코드가 성공적으로 실행되지 않을 수 있다. 또한 프로그램 코드를 좀 더 복잡하게 만들 수 있다. 즉, 대소문자를 구분하여 작성하게 되면 식별자 간의 차이가 작고 감지하기 어렵기 때문에 프로그래머가 프로그램을 읽기 어렵고 오류가 발생할 수 있다.

**18. Many contemporary languages allow two kinds of comments: one in which delimiters are used on both ends (multiple-line comments), and one in which a delimiter marks only the beginning of the comment (one- line comments). Discuss the advantages and disadvantages of each of these with respect to our criteria**.

많은 프로그래밍 언어는 두 가지의 타입의 주석이 존재한다. 하나는 행 단위 주석이고 다른 하나는 블록 단위의 주석이다. 두 주석 모두 장단점이 있으므로 실제로 두가지 종류의 주석이 모두 필요에 따라 적용된다. 이 중 어떤 것이 좋은지에 대해서는 프로그래밍을 할 때 상황에 맞춰서 사용하면 된다. 행 단위 주석은 한 줄 이하의 작은 설명이 필요한 상황에서 더 적합하다. 이러한 유형의 주석은 일반적으로 특정 변수가 무엇을 보유하고 있는 지와 같이 줄에 변수 선언 후 다른 것에 대한 정보를 제공하는데 사용된다. 따라서, 구분 기호를 지정하고 명령문 뒤에 주석을 계속 추가하는 것이 쉽다. 블록단위 주석의 경우 프로그램 시작 시 알고리즘에 대한 정보를 제공하기 위해 단락 크기의 주석을 사용하고 프로그램에 사용된 기능과 변수에 대한 요약을 제공하는 경우에 선호된다. 이 경우 양쪽 끝에 구분 기호가 필요하다. 주석이 끝날 때, 구분 기호가 제공되지 않은 상태에서 시작 구분 기호가 없는 다음 줄로 이어지게 되면 코드가 구문 상 불규칙하게 될 수 있다.

**2. 프로그래밍 언어의 평가 기준 4가지가 무엇인지 각각 15줄 정도로 알기 쉽게 요약하시오**

프로그래밍 언어의 평가 기준에는 판독성, 작성력, 신뢰성, 비용이 있다.

**판독성**

프로그래밍 언어를 평가하는 데 있어서 중요한 기준 중의 하나는 프로그램을 얼마나 쉽게 읽고 이해할 수 있느냐는 것이다. 판독성에 기여하는 특성들로는 전반적인 단순성, 직교성, 데이터 타입, 구문설계가 있다. 첫 번째 특성으로는 단순성이 있다. 많은 기본 구조를 가진 언어는 적은 개수의 기본 구조를 가진 언어보다 더 배우기 어렵다. 물론, 언어의 단순성이 지나치게 이루어질 수 있고 그 예시로는 어셈블리 프로그램이 있다. 어셈블리 프로그램은 문장들이 단순하기 때문에, 동일한 프로그램을 고급 언어로 작성했을 경우보다 더 많은 문장들이 필요하게 된다. 두 번째 특성으로는 직교성이 있다. 프로그래밍 언어에서 직교성이란 상대적으로 적은 개수의 기본 구조들이 적은 개수의 방법으로 조합되어 언어의 제어 구조와 데이터 구조가 생성될 수 있다는 것을 의미한다. 직교성은 언어의 기본 요소들 간의 관계 대칭성을 의미하는데, 직교성이 결여되면 언어 규칙에서 예외 사항이 초래된다. 지나치게 높은 직교성도 불필요한 복잡성이라는 문제를 초래하게 되는데, 언어가 많은 기본 요소들을 포함하기 때문에 조합이 단순할지라도, 상당수의 조합은 복잡성을 초래한다. 세 번째 특징으로는 데이터 타입이 있다. 언어가 데이터 타입과 데이터 구조를 정의할 수 있는 충분한 장치를 제공하면 판독성에 또 다른 큰 도움이 된다. 마지막 특성은 구문 설계이다. 언어의 요소들에 대한 구문, 즉 형식은 프로그램 판독성에 영향을 미친다. 프로그램 형태와 판독성은 언어의 특수어의 형식에 의해 상당한 영향을 받는다. 제어 구조에서 주로 복합문, 문장들의 그룹을 형식화하는 방법이 특히 중요하다. 문장의 형태가 부분적으로 그 목적을 나타낼 수 있도록 문장을 설계하는 것 또한 판독성에 도움을 준다. 의미는 구문, 즉 형식으로부터 직접 파악되어야 한다. 이런 원칙은 두 개의 언어 구조가 그 외형에 있어서는 동일하거나 유사해 보이나 문맥에 따라 다른 의미를 가질 때 위반된다.

**작성력**

작성력은 선택된 문제 영역에 대해 프로그램을 생성하는 데 언어가 얼마나 쉽게 사용될 수 있는가에 대한 척도이다. 판독성에 영향을 미치는 언어 특성의 대부분은 작성력에도 영향을 미친다. 판독석의 경우처럼, 작성력도 언어의 목표 문제 영역의 문맥에서 고려되어야 한다. 한 언어가 특정 응용 분야에 대해서 설계되었고, 다른 언어는 그렇지 않았을 때, 이 특정 응용 분야의 영역에서 두 언어의 작성력을 비교하는 것은 타당하지 않다. 언어의 작성력에 영향을 미치는 중요한 특성들로는 단순성과 직교성, 추상화의 지원, 표현력이 있다. 첫 번째 특성으로는 단순성과 직교성이 있다. 언어가 상당히 많은 구조를 포함하면, 어떤 프로그래머는 이런 구조 모두를 잘 알고 있지 않을 수 있다. 따라서 적은 수의 기본 구조와 이들을 조합하기 위한 일관된 규칙(직교성)을 갖는 것이 단순히 많은 기본 구조를 갖는 경우보다 훨씬 좋다. 한편, 지나치게 높은 직교성은 작성력에 해가 될 수 있다. 기본 구조들에 대한 거의 모든 조합이 적법할 때, 프로그램의 오류가 탐지되지 않을 수 있다. 이는 컴파일러가 탐지할 수 없는 이해하기 어려운 코드를 초래할 수 있음을 의미한다. 두 번째 특성으로는 추상화의 지원이 있다. 추상화는 많은 세부 사항이 무시될 수 있는 방식으로 복잡한 데이터 구조나 연산을 정의하여 사용할 수 있는 능력을 의미한다. 프로그래밍 언어는 두 가지 종류의 추상화 프로세스와 데이터를 지원할 수 있다. 세 번째 특성으로는 표현력이 있다. 표현력은 여러 가지 특성으로 기술될 수 있다. APL과 같은 언어에서, 표현력은 상당한 양의 계산을 작은 프로그램으로 수행 가능하게 하는 강력한 연산자들이 존재한다는 것을 의미한다. 즉, 언어가 계산을 명세하는 데 있어서 상대적으로 편리한 방식을 제공한다는 것을 의미한다.

**신뢰성**

프로그램은 모든 조건 하에서 주어진 명세에 따라 수행한다면 신뢰적이라고 말한다. 신뢰성에 중요한 영향을 미치는 여러 가지 언어 특징으로는 타입 검사, 예외 처리, 별칭, 판독성과 작성력이 있다. 첫 번째 특징은 타입 검사이다. 타입 검사는 단순히 컴파일러에 의해서나 프로그램 실행 중에 주어진 프로그램에 포함된 타입 오류에 대한 검사이다. 프로그램의 오류를 더 이른 시간에 탐지할수록, 요구된 복구 비용이 더 적게 든다. 두 번째 특성으로는 예외처리가 있다. 프로그램이 실행-시간 오류(프로그램에서 탐지 가능한 다른 비정상적인 조건들을 포함하여)를 가로채서, 이를 올바르게 수정한 후에 계속 실행할 수 있도록 하는 능력이다. Ada, C++, Java, C#은 예외 처리를 위한 확장된 능력을 갖는다. 그러나 이런 기능은 C와 Fortran을 포함하여 널리 사용되고 있는 많은 언어에서 실질적으로 지원되지 않고 있다. 세 번째 특징은 별칭이다. 별칭은, 간단히 정의하면,동일한 기억 장소에 접근하는 데 사용할 수 있는 두 개 또는 그 이상의 다른 이름을 갖는 것이다. 대부분의 프로그래밍 언어는 별칭의 어떤 유형을 허용한다. 동일한 변수를 가리키도록 설정된 두 개의 포인터가 한 예제이다. 이런 별칭은 대부분의 언어에서 가능하다. 이런 프로그램에서, 프로그래머는 두 개 중에서 한 개의 포인터가 가리키는 값을 변경하면 다른 포인터에 의해 참조되는 값이 변경된다는 사실을 항상 기억해야 한다. 어떤 언어에서는 부족한 데이터 추상화 기능을 보완하기 위해서 별칭을 사용하며, 다른 언어에서는 언어의 신뢰성을 향상시키기 위해서 별칭을 제약한다. 마지막 특징으로는 판독성과 작성력이다. 요구된 알고리즘을 표현하는 자연스러운 방법을 지원하지 않는 언어로 작성된 프로그램은 부자연스러운 접근 방법을 사용하게 될 것이다. 이러한 방법은 모든 가능한 상황에 대해 덜 올바를 가능성이 많다. 판독성은 소프트웨어 생명 주기의 코딩과 유지보수 단계에서 신뢰성에 영향을 미친다. 판독하기 어려운 프로그램은 작성하고 수정하기에 어렵다.

**비용**

프로그래밍 언어의 최종 비용은 언어의 여러 가지 특성들로 결정된다. 첫째, 언어를 사용하는 프로그래머를 교육시키는 비용이다. 이런 비용은 언어의 단순성과 직교성 그리고 프로그래머의 경험에 의해 결정된다. 둘째, 언어를 사용하여 프로그램을 작성하는 비용이다. 이런 비용은 언어의 작성력에 의해 결정되는데, 프로그램의 목적에 언어가 얼마나 근접하고 있는가에 부분적으로 따른다. 셋째, 언어로 작성된 프로그램을 컴파일하는 비용이다. 넷째, 프로그램을 실행시키는 비용은 언어의 설계에 의해 상당히 많은 영향을 받는다. 많은 실행-시간 타입 검사를 요구하는 언어는 컴파일러의 성능에 관계없이 빠른 코드 실행을 어렵게 할 것이다. 최적화는 컴파일러가 생성하는 코드의 크기를 줄이거나 실행 속도를 향상시키기 위해서 사용될 수 있는 기법들의 모임에 부여된 이름이다. 최적화가 거의 또는 전혀 수행되지 않은 경우에, 컴파일은 최적화된 코드를 생성하기 위해서 많은 노력이 투여된 경우에 비해서 더 빠르게 수행될 수 있다. 이는 컴파일러가 사용되는 환경에 따라서 달라진다. 다섯 번째는 언어 구현 시스템의 비용이다. 구현 시스템이 비싸거나 단지 비싼 하드웨어에서만 실행되는 언어는 널리 사용될 수 있는 기회가 더 적을 것이다. 여섯 째, 신뢰성 부족에 따른 비용이다. 위험하지 않은 시스템의 실패도 상실된 미래 사업이나 결함 소프트웨어 시스템에 대한 소송의 관점에서 볼 때 비용이 높을 수 있다. 마지막으로, 프로그램을 유지보수하는 비용이다. 코드 교정과 새로운 기능을 추가하는 수정 작업이 포함된다. 언어 비용에 기여하는 모든 요소 중에서 프로그램 개발, 유지보수, 신뢰성이 제일 중요하다. 이런 항목들은 작성력과 판독성에 의해서 결정되므로, 이 두 가지 평가 기준이 가장 중요하다. 대부분의 평가 기준, 특히 판독성, 작성력과 신뢰성은 정확하게 정의되어 있지 않으며 정밀하게 측정될 수 없다. 그럼에도 이런 평가 기준은 유용한 개념이며, 프로그래밍 언어의 설계와 평가에 있어서 가치 있는 통찰력을 제공한다.

**3. C언어와 같은 고급 언어의 프로그램을 컴파일러 혹은 인터프리터를 사용하여 실행하는 경우를 비교하여 서로의 장단점을 20줄 정도로 설명하시오.**

프로그램은 컴퓨터에서 직접 실행될 수 있는 기계어로 번역될 수 있는데 이러한 방법이 컴파일러 구현이라고 불린다. 컴파일러는 번역 과정이 완료되면 매우 빠른 프로그램 실행의 장점을 갖는다. C, COBOL, C++, Ada와 같은 언어의 대부분 상용적 구현은 컴파일러 방법으로 이루어진다. 컴파일러의 선택사항으로 프로그램의 크기를 줄이거나 실행 속도를 더 빠르게 함으로써(보통 중간 코드 버전의) 프로그램을 향상시키는 최적화가 있다. 어떤 컴파일러는 어떠한 의미 있는 최적화도 수행할 수 없다. 이런 유형의 컴파일러는 번역된 프로그램의 실행 속도가 컴파일 속도보다 훨씬 더 중요하지 않은 상황에서 사용된다. 대부분의 상업과 산업의 현장에서, 실행 속도는 컴파일 속도보다 더 중요하므로 최적하는 통례적으로 바람직하다. 많은 종류의 최적화를 기계어 상에서 수행하기가 어려워서 대부분의 최적화는 중간 코드 상에서 이루어진다.

순수 해석은 구현 방법에서 컴파일과 정반대에 위치한다. 이 방법에서, 프로그램은 어떠한 번역 과정 없이 인터프리터라 불리는 또 다른 프로그램에 의해 해석된다. 인터프리터 프로그램은 기계에 대한 소프트웨어 모의실험으로서 동작한다. 이 경우 인출-실행 사이클은 기계 명령어가 아닌 고급 언어 프로그램 문장을 다룬다. 순수 해석은 많은 원시-수준 디버깅 연산을 쉽게 구현할 수 있는 장점을 갖는다. 이는 모든 실행-시간 오류 메시지가 원시-수준 프로그램 단위를 참조할 수 있기 때문이다. 반면, 이 방법은 실행 시간이 컴파일된 시스템보다 10배 내지 100배 정도 느린 심각한 단점을 갖는다. 이렇게 실행 속도가 느린 주요 요인은 고급 언어 문장의 해석에 있다. 이런 해석은 기계어 명령어보다 훨씬 더 복잡하다. 더욱이, 문장이 몇 번 실행되는지 상관없이, 그 문장은 매번 해석되어야 한다. 순수 인터프리터의 다른 단점은 흔히 더 많은 기억 공간을 요구한다는 것이다. 원시 프로그램 외에도 심볼 테이블이 해석 과정에서 존재해야 한다. 더욱이 원시 프로그램은 최소의 크기로 제공하는 형식이기보다는 접근과 수정이 용이하도록 설계된 형식으로 저장될 수 있다. 1960년대 초반의 단순한 언어들(APL, SNOBOL, LISP)이 순수 해석으로 구현되었지만, 1980년대에는 이 방법이 고급 언어에 대해 거의 사용되지 않았다. 그러나 최근에 순수 해석이 JavaScript와 PHP같은 Web 스크립트 언어에서 널리 사용되고 있다.

**4. 과학계산용으로 FORTRAN이나 C언어, 비즈니스 업무용으로는 COBOL같은 프로그래밍 언어가 구별되어 사용되는 이유가 무엇인지 설명하시오.**

FORTRAN과 C언어 등 과학 계산용 언어는 복잡한 계산이나 수식 등이 포함된 프로그램을 처리할 수 있다. 문법이 비교적 간단하고 호환성이 높으며 문법이 간단하고 계속 개발되는 범용의 고급 언어로 사용되고 있다. 가장 큰 특징은 복잡한 수학 연산식과 같은 순서의 표현을 그대로 쓰기 때문에 프로그램 작성이 간단하면서도 매우 쉽다. 특히, 문법이 비교적 간단하여 단기간 내에 습득이 가능하다는 장점이 있다. 단점으로는 어셈블리 언어처럼 다양한 자료 처리가 불가능하고 사무용 자료 처리의 문제와 결과 출력 상에 제약이 많아 사무 처리용 언어로는 적합하지 못하다.

COBOL은 사무처리에 적합한 언어로 현재 널리 사용되고 있는 고급언어이다. 컴퓨터의 기종에 관계없이 공통적으로 사용되며 일상적으로 사용하는 영어 문장을 사용하므로 읽고, 쓰기가 쉽다. 또한, 프로그램의 수정 및 오류 검색이 쉽다. 문서 기능을 갖추고 있고 프로그램이 4개의 Division으로 구성되며 정형화된 형식을 취하고 있어 문서 그 자체를 보고서로 사용할 수 있다. 파일 처리가 뛰어나며 작성이 용이하고 작성된 프로그램이 다른 사람에게도 쉽게 이해될 수 있다. 단, 컴파일러에는 번역을 위한 많은 항목을 포함하고 있어 이의 수용을 위한 주 기억장치의 용량이 커져야 한다. 또한 프로그램 작성량이 많고 길어서 전체적인 구성이 간결하지 못하다는 단점이 있다.

**5. C언어를 사용하여 주어지는 입력되는 년도의 달력을 한 페이지에 출력하는 프로그램을 작성하고 그 실행 결과를 스크린샷 포함하여 같이 제출하시오. 달력 프로그램에서는 array[12][42]와 같이 2차원 혹은 3차원 배열을 이용하여 각 원소에 들어갈 날짜를 계산하여 저장한 다음에 이 배열을 출력하도록 프로그램을 작성하도록 한다.**

**1) 실행 코드**

//

// Calendar\_array.c

// Project\_PL

//

// Created by yerimhuh on 2020/03/21.

// Copyright © 2020 yerimhuh. All rights reserved.

//

#include <stdio.h>

**int** basemonth[] = {31,28,31,30,31,30,31,31,30,31,30,31};

**int** firstday;

**int** year;

**int** leapyear(**int** a){

**if**(a % 4 == 0){

**if**(a % 100 == 0){

**if**(a % 400 == 0){

**return** 1;

}

**return** 0;

}

**return** 1;

}

**return** 0;

} // 입력받은 년도가 윤년인지 검사하는 함수

**int** first\_day(**int** i){

**if**((i == 0)|(i == 2)| (i == 4) | (i == 6) | (i == 7) |( i == 9) | (i == 11))

firstday = (firstday + 31) % 7;

**else** **if**((i == 3) | ( i == 5)| (i == 8) | (i == 10))

firstday = (firstday + 30) % 7;

**else** **if**(i == 1){

**if**(leapyear(year))

firstday = (firstday + 29)% 7;

**else**

firstday = (firstday + 28) % 7;

}

**return** firstday;

}//매 달 1일이 시작하는 요일을 정해주는 함수

**void** calendar(**int** year){

**int** inputyear;

**int** leapcnt=0;

**int** month[12][6][7]= {0};

**int** i,j,k,l;

**int** daycnt;

**for**(k = 1; k < year; k++){

leapcnt += leapyear(k);

}// 입력받은 년도까지 윤년이 몇 번인지 저장한다.

inputyear = year-1;

firstday = (inputyear\*365+leapcnt+1)%7; //1월1일이 무슨 요일인지 정해주는 식

**for**(i = 0; i < 12; i++){

daycnt = 0;

**for**(j = 0; j < 6; j++){

**for**(k = 0; k < 7; k++){

daycnt++;

month[i][j][k] = daycnt-firstday;

**if**(month[i][j][k] < 1 || month[i][j][k] > basemonth[i])

month[i][j][k] = 0;

}

}

firstday = first\_day(i);

} // 3차원 배열에 날짜를 저장해준다.

**for**(l = 0; l < 4; l++){

printf("--------------%d월-------------\t--------------%d월-------------\t--------------%d월-------------\n",l\*3+1,l\*3+2,l\*3+3);

printf(" 일 월 화 수 목 금 토 \t 일 월 화 수 목 금 토 \t 일 월 화 수 목 금 토 \n");

printf("-------------------------------\t-------------------------------\t-------------------------------\n");

**for**(i = 0; i < 6; i++){

**for**(j = 0; j < 3; j++){

**for**(k = 0; k < 7; k++){

**if**(month[(l\*3)+j][i][k] == 0){

printf(" ");

**continue**;

}

printf("%3d ",month[(l\*3)+j][i][k]);

}

printf("\t");

}

printf("\n");

}

printf("\n\n");

}

}

**int** main(){

**while**(1){

printf("년도를 입력하세요 : ");

scanf("%d",&year);

**if**(leapyear(year) == 1)

basemonth[1] = 29;

calendar(year);

}

}

**2) 실행 화면 캡쳐**

**점수판이(가) 표시된 사진

자동 생성된 설명**