**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?

프로그래밍을 할 때 있어서 단일 언어를 사용하면 프로그램을 유지 및 보수하는 것에 있어서 편리합니다. 개인적으로 느낀 바로는 여러가지 프로그래밍 언어로 연결된 프로그램들 보다는 단일 언어로 작성된 프로그램이 전체적인 구성과 흐름, 그리고 코드를 작성 할 때에 호환성을 크게 걱정하지 않아도 된다는 장점이 있습니다. 또한 이로 인해 하드웨어도 해당 언어에 적합하게 발전될 가능성이 큽니다. 게다가 전세계의 사람들이 단일언어로 사용함으로서 발생하는 수많은 문서와 지식은 기술의 범용성을 높여줄 것이 당연하다고 생각됩니다.

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

그러나 개인적으로는 프로그래밍을 할 때 단일 언어를 사용하는 것은 옳지 못하다고 생각합니다. 물론 단일 언어로 사용함으로서 나타나는 이점은 앞서 말했다시피 누구나 프로그램을 개발하는 것에 있어서 접근이 쉽고, 코드를 수정 혹은 필요에 의해 재가공할 때 이해하기에 편리하다는 장점이 있습니다. 그러나 문화에 유행이 존재하듯, 프로그래밍 언어도 마찬가지라고 생각합니다. 당장 우리가 사용하는 컴퓨터 하나만으로도 기계어, 어셈블리어, 고급언어로 나뉩니다. 제가 생각하는 이 언어들의 차이 중 가장 큰 부분은 판독성이라고 생각합니다. 사람마다 느끼는 코드의 판독 능력이 다르기 때문에 이러한 구분이 생겼고, 고급 언어에서는 더 편리한 코드의 이해를 하기 위해서 판독의 유행을 따라가는 부분이 크게 작용하는 것 같습니다.

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

이러한 작성 요령의 장점은 코드를 작성하고 이해할 때에 크게 드러납니다. 변수와 함수의 경우에 우리는 대문자를 구분 지으며 핵심 기능과 보조 기능을 명확하게 구분할 수 있습니다. 다시 말해서 보다 코드를 정확하게 읽게 해줄 수 있으며, 코드의 혼동을 방지할 수 있습니다. 그러나 대소문자 구분으로 인해 컴파일러 단계에서 이를 구분하게 해주는 알고리즘을 추가해야 하며, 동일한 알파벳이나 대소문자 구분으로 다른 기능을 하는 변수 혹은 함수일 경우에 코드를 난독하게 할 수 있습니다.

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.

주석은 프로그래밍을 할 때 상황에 맞춰서 사용하면 된다고 생각합니다. 문장의 앞에 작성하는 주석의 경우에는 간단한 구문의 기능 혹은 이 코드가 사용되어야 하는 이유에 대해서 간략하게 서술할 때 주로 사용합니다. 명령문 전후로 빠르게 주석을 작성하는것이 용이하나 이를 많이 사용하면 코드의 가독성에 영향을 미칠 수 있습니다. 단락 주석의 경우에는 함수의 pre, post condition등의 정보를 제공할 때 사용하며, 글쓰기로 비유했을 때 일종의 서론에 해당할 수 있습니다. 이것은 많은 량의 정보를 한번에 서술할 수 있습니다. 그러나 구문별로 서술하는데에 적합하지 않습니다. 이 또한 가독성에 영향을 미칠 수 있습니다.

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

평가기준: 가독성, 작성력, 신뢰성, 비용

우리는 프로그래밍 언어를 사용함에 있어서 코드를 읽기 쉽게 볼 수 있도록 작성한다. 이러한 기준점이 되는 것이 가독성. 이러한 가독성을 좋게 하기 위해서 언어가 단순하게 작성될 수 있어야 한다.

언어를 복잡하게 만드는 이유 중 첫번째가 다중성인데, 대표적인 예로 a++, ++a, a += 1, a = a+1 이다. 조금씩 의미에 차이는 존재하지만 결국 변수를 1만큼 증가시킨다. 이러한 다양성을 가지면 많은 경우를 만들어 내고, 가독성을 감소시키는 이유가 된다.

두번째로 직교성이다. 작은 요소들이 합쳐져서 제어의 기능과 데이터 구조가 생성된다는 것을 일컫는다. 대표적으로 포인터변수를 예로 들 수 있으며, 이러한 특징은 프로그램 형태와는 관련이 없다. 그러나 기본 요소들간의 관계 대칭성을 의미하기도 하는데, 직교성이 결여되면 언어를 사용함에 있어서 예외가 발생할 수 있다. 결국 단순성과 직교성은 밀접하게 관련되어있는데, 언어 설계가 직교적이고 단순할 수록 더 적은 예외를 갖고 결국 깔끔한 코드가 나오게 된다.

세번째로 데이터타입과 구조이다. 언어가 자료 타입과 자료 구조를 정의할 수 있는 충분한 장치를 제공하면 가독성에 도움이 된다. 대표적인 예로 bool타입이 없어서 참과 거짓을 0과 1로 표현하는 경우이다.

마지막으로 구문설계이다. 제어구조에서 복합문을 형식화 하는 방법에서 어떤 언어는 특수어나 기호들로 구성된 서로 결합되는 쌍을 이용해 문장 그룹을 형식화 한다. { } 와 같은 대괄호 형식으로 형식화한다. 포트란 95와 ada에서는 end if , end loop와 같은 형식을 취하기도 한다. 그리고 문장의 뜻은 문맥에 따라 다른 의미를 가지면 안된다. 대표적으로 c언어의 static은 함수 내부에서는 변수 정의에 사용되면 컴파일 시간에 생성됨을 의미하고 함수 밖에선 그 정의가 나타난 파일 내부에서만 작동함을 의미하므로 문맥에 따라 의미가 달라진다.

두번째 기준점은 작성력이다. 작성력은 선택된 문제 영역에 대해 프로그램을 생성하는데 언어가 얼마나 쉽게 사용될 수 있는가에 대한 척도이다. 즉 잘 작성 가능한가에 대한 확인이다. 이것은 크게 세가지로 판단할 수 있다.

먼저 이러한 작성력을 평가하기 위해서는 위에서 언급한 직교성과 다중성의 반대인 단순성을 확인하는 것이다. 적은 수의 기본구조를 통하여 복잡한 문제를 해결할 수 있는지에 대한 확인과 직교성에 지나치게 의존하는지에 대한 확인을 하면 된다.

두번째로 추상화의 지원이다. 많은 세부 사항들을 작성하지 않더라도 큰 틀을 잡아 두어 복잡한 데이터 구조나 연산을 정의하여 사용할 수 있다. 즉 많은 세부사항이 무시될 수 있는 방식으로 복잡한 데이터 구조나 연산을 정의하고 사용할 수 있는 능력을 의미한다. 이러한 추상화에는 데이터(변수, 배열, 구조체), 프로세스(subprogram), 그리고 데이터 관련 연산들을 통합하여 표현한 클래스 등이 있다.,

마지막으로 표현력이다. 단순히 많이 표현하는 것을 뜻하는 것이 아닌, 일반적으로 제공된 표현보다 더욱 단순한 표현이 가능하게 만드는 표현이 존재하는가에 대한 문제이다. 즉 표현력이란 언어가 계산을 명세하는데 있어서 상대적으로 알아보기 쉬우면서도 편리한 작성을 지원하는 것이다. 앞서 언급한 I = i+1을 i++로 표현가능한 것과 같은 의미이다. 또한 이외에도 case구문과 같은 표현이 없다면 if-then-else를 여러 번 사용해야 한다.

세번째 기준점은 신뢰성이다. 프로그램은 모든 조건에서 주어진 명세에 따라 수행한다면 신뢰적이라고 한다. 프로그램의 오류를 일찍 탐지할수록 시간비용을 줄일 수 있다. 이러한 신뢰성을 판단하기 위한 기준은 3가지가 존재한다.

먼저 타입검사이다. 적절하지 않은 타입의 사용이나 서로 다른 타입의 연산이 있는지 검사할 수 있는 지의 여부를 확인한다. 타입검사는 언어 신뢰성에 중요한 요소로 c에서는 gcc할 때 오류를 찍어서 내가 항상 선언한 매개변수의 형과 개수를 main안에서 사용할 때 잘못 사용해도 알지 못한다. 그러나 java는 중간중간에 계속 !기호를 알려주므로 미리 확인할 수 있다.

두번째로 예외처리이다. 프로그램 실행간 발생한 오류를 실시간으로 수정하여 지속적으로 프로그램이 돌아가는가에 대한 여부를 확인한다.

마지막으로 별칭이다. 동일한 기억장소에 대해서 2개 이상의 다른 참조방법을 갖는 것을 의미한다.

마지막 기준점은 비용이다. 프로그래밍 언어의 최종 비용으로 첫번째 요소는 프로그래머 교육비용으로 언어의 단순성과 직교성 그리고 프로그래머의 경험에 의해 결정된다.

둘째, 프로그램 작성비용이다. 이런 비용은 언어의 작성력에 의해 결정되는데, 이는 특정 응용 프로그램의 목적에 언어가 얼마나 근접하고 있는가에 부분적으로 따른다.

셋째, 프로그램 컴파일 비용이다.

넷째, 프로그램 실행 비용이다. 프로그램을 실행시키는 비용은 언어의 설계에 의해 상당히 많은 영향을 받는다. 많은 실행-시간 타입 검사를 요구하는 언어는 컴파일러의 성능에 관계없이 빠른 코드 실행을 어렵게 할것이다.

다섯째, 언어 구현 비용이다. 구현시스템이 비싸거나 단지 비싼 하드웨어에서만 실행되는 언어는 널리 사용될 수 있는 기회가 훨씬 더 적을 것이다.

여섯째, 신뢰성 부족에 따른 비용이다. 원자력 발전소나 의료용 엑스레이와 같은 매우 위험한 시스템에서 소프트웨어가 실패한다면, 그 비용은 상당히 높을 수 있다.

마지막으로, 프로그램 유지 보수 비용이다. 이중에서 가장 중요한 요소는 유지보수 비용이다. 상대적으로 긴 생명주기를 갖는 대형 소프트웨어의 경우에, 유지보수 비용이 개발비용의 2배에서 4배까지 높을 수 있다고 알려져 있다.

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

먼저 인터프리터를 사용하여 실행하는 경우에는 인터프리터만 설치되어 있으면 같은 프로그램을 다른 컴퓨터에서도 실행시킬 수 있다. 물론 인터프리터가 컴퓨터마다 다르게 컴파일 되어야 한다는 경우의 수가 존재하지만, 실행하는 환경에 맞는 인터프리터가 있다고 가정을 한다. 그러나 인터프리터가 각 구문마다 번역 후 실행의 과정을 거쳐야 하므로 비교적 컴파일러로 번역 후 실행한 프로그램에 비해서 속도가 느리다. Int sum = i+j; printf(“%d”, sum); if(sum > 10)과 같은 구문이 있으면 세미콜론이 붙은 구문마다 컴파일 후 실행을 반복한다. 이에 비해서 컴파일러는 바이너리를 실행시키는 것이기 때문에 실행속도가 인터프리터보다 비교적 빠르다. 컴파일러는 한 언어에서 다른 언어로 번역하는 프로그램을 말한다. 컴파일러는 우리가 작성한 프로그램을 한꺼번에 기계어로 번역한다. 즉 프로그램을 작성하면 컴파일 후 실행을 하는 구조이다. 인터프리터와 번역 후 실행이라는 구조는 같지만 실행 과정에서 차이가 존재한다. 그러나 한번에 번역하고 실행하기 때문에 앞서 말한 결과가 도출된다. 그러나 컴파일러에도 실행 환경에 맞게 명령어 셋이 다를 수 있다. 따라서 한 컴퓨터에서 컴파일 해서 실행한 프로그램은 명령어 셋이 다른 컴퓨터에서 동작하지 않을 수 있다. 결국 명령어 셋이 다른 컴퓨터마다 컴파일을 새로 해주어야 한다. 이러한 일례로 윈도우 운영체제에서 x86과 x64 즉 32비트와 64비트 환경에 맞는 프로그램을 다운받는 것이다. 이렇게 인터프리터가 번역해야 하는 프로그래밍 언어들을 스크립팅 언어라고 부르며 대표적으로 javascript, 파이썬, php등이 이에 해당된다. 반대로 컴파일러에 해당하는 언어들은 우리가 아는 C, C++등이 해당된다.

**4. 과학 계산용으로 FORTRAN이나 C언어, 비즈니스 업무용으로는 COBOL과 같은 프로그래밍 언어가 구별되어 사용되는 이유가 무엇인지 설명하시오.**

먼저 포트란이 개발될 당시 주변환경의 컴퓨터는 메모리 용량이 작고, 느리며, 상대적으로 신뢰성이 없었다. 그리고 컴퓨터의 주요 사용처가 과학 계산분야를 위한 것이었고, 컴퓨터를 프로그래밍하기 위한 달리 존재하는 효율적이고 효과적인 방법이 없었다. 마지막으로 프로그래머의 비용과 비교해서 컴퓨터의 고비용 때문에 생성된 목적 코드의 속도향상은 첫번째 포트란 컴파일러의 주요 목적이었다. 즉 포트란의 경우 태생부터 목적이 분명했고, 효율성이 뛰어났다. 컴파일러에 의해 생성되는 기계코드가 직접코딩으로 생성할 수 있는 코드의 효율을 어느정도 대체했기 때문이다. 이후 변수에 대한 명시적 타입선언, 논리 IF문 등 현재의 고급언어의 기반이 되는 기능들의 추가로 그 당시에 널리 사용되는 프로그래밍 언어들 중 하나가 되었다. 그리고 연산을 하는 모든 업무에 관하여 영향력을 끼치게 되었다.

반면에 코볼의 경우 사무 기록의 전산화를 목적으로 개발되었다. 가장 중요한 이유중에 하나는 이 언어가 개발된 이후 사무 응용을 위한 새로운 언어를 설계하려는 시도가 거의 없었다. 이는 코볼의 기능들이 그 응용영역의 필요성을 얼마나 잘 충족하였는가에 부분적으로 기인한다. 그리고 지난 30년동안 사무 컴퓨팅에서의 상당한 규모의 성장이 규모가 작은 사무에서 일어났다는 것이다. 이러한 사무에서는 소프트웨어 개발이 거의 이루어지지 않고, 사용된 대부분의 소프트웨어에서는 다양한 일반적인 사무 응용을 위한 기성품 패키지 형태로 판매되었다. 코볼은 매크로들을 편리하게 제공해주는 언어였고, 미 국방성에서 공식적으로 지시된 언어였고, 이를 통해 신뢰성까지 확보했다.

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

소스코드 링크: <https://github.com/Codejune/C_Calendar>

|  |
| --- |
| #pragma warning(disable:4996)  #include <stdio.h>  #include <string.h>  #define TRUE 1  #define FALSE 0  #define NONE -1  int arr[12][6][7] = { 0 };  int basicyear[12] = { 31,28,31,30,31,30,31,31,30,31,30,31 };  int isLeap(int year) {  /\*  1) 4년으로 나뉘면 윤달(2/29)존재.  2) 그러나 100년으로도 나뉘면 윤달 없음.  3) 또한 400년으로 나뉘면 윤달 존재  \*/  if ((year % 4 == 0) && !(year % 100 == 0) || (year % 400 == 0))  return TRUE;  else return FALSE;  }  int countDay(int year) {  int sum = 365;  int i;  for (i = 1; i < year; i++)  if (isLeap(i))  sum += 366;  else sum += 365;  return sum;  }  void assignCalendar(int year, int sum) {  int i, j, k;  int none\_count = -1;  int total\_count = 0;  // 1월1일 무슨 요일인지 계산  k = sum % 7;  //시작 요일에 맞춰 정렬  for (i = 0; i < 12; i++) {  for (j = 0; j < k; j++)  arr[i][0][j] = NONE;  if(isLeap(year) && i == 1)  sum += 29;  else sum += basicyear[i];  k = sum % 7;  }  for (i = 0; i < 12; i++) {  for (j = 0; j < 6; j++) {  for (k = 0; k < 7; k++) {  if (isLeap(year) && (total\_count - none\_count) > 29 && i == 1) break;  if (!isLeap(year) && (total\_count - none\_count) > basicyear[i] || (total\_count - none\_count) > basicyear[i] && i != 1) break;  if (arr[i][j][k] == NONE) {  none\_count++;  } else {  arr[i][j][k] = total\_count - none\_count;  }  total\_count++;  }  }  total\_count = 0;  none\_count = -1;  }  }  int printCalendar(int year) {  int i, j, k, m;  int month = 1;  int day = 0;  for (i = 0; i < 3; i++) {  for(j = 0; j < 4; j++)  printf(" %d ", month++);  printf("\n");  for (j = 0; j < 4; j++)  printf("SUN MON TUS WED THU FRI SAT ");  printf("\n");  for (j = 0; j < 4; j++)  printf("=================================================== ");  printf("\n");  for (j = 0; j < 6; j++){  for (k = (i\*4); k < i \* 4 + 4; k++) {  for (m = 0; m < 7; m++) {  if (arr[k][j][m] == NONE || arr[k][j][m] == 0) {  if (m == 6) printf(" ");  else printf(" ");  continue;  } else {  if ((arr[k][j][m] / 10) > 0)  printf(" ");  else printf(" ");  printf("%d", arr[k][j][m]);  if(m != 6)  printf(" ");  }  }  printf(" ");  }  printf("\n");  }  printf("\n\n");  }  memset(arr, 0, sizeof(arr));  }  int main(void) {  int year;  while (1) {  printf("Programming Language assignment #1(Exit : -1)\n");  printf("Year (ex.2020): ");  scanf("%d", &year);  if (year == NONE) break;  assignCalendar(year, countDay(year));  printCalendar(year);  }  return 0;  } |

[코드 1] arr\_calendar.c

실행 결과

 