BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
PILANI CAMPUS
FIRST SEMESTER 2020 – 2021
PRINCIPLES OF PROGRAMMING LANGUAGES (CS F301)
TUTORIAL 1 (28TH AUGUST 2020)

---

1. Analyze the pros and cons of pure compilation process with pure interpretation process? See with examples of both. Do the same analysis with hybrid compilation and Just in Time (JIT) compilation. What do you observe with most famous examples of all four languages with respect to the time in compilation, the debugging environment, the learning, the portability, etc. Does it go in line with the ambitions of the languages designed?
2. Java does strict checking for all the references in the array for bound checking. It does other checks at the time of compilation. Evaluate JAVA with respect to this feature against C language which does not check bounds effectively during compile time.
3. **APL** has a rich set of built-in functions (and "operators" that are applied to functions to yield different functions) that operate on scalar, vectors, arrays, even higher-dimensional objects, and combinations of them. Here are some tables from the 1970 "APL360 User's Manual" [3] that give a flavor of the power and sophistication of the built-in APL functions and operators.

   APL example:

   (~T∈T∘.×T)/T←1↓ιR   /* A line of code in APL*/

   Here is how this expression is evaluated:

subexpression	**meaning**	**value if R is 6**
ιR	Generate a vector of numbers from 1 to R.	1 2 3 4 5 6
T←1↓	Drop the first element of the vector and assign the rest to the temporary vector T.	2 3 4 5 6
T∘.×T	Create the multiplication outer product: a table that holds the result of multiplying each element of T by each element of T.	$\begin{array}{ccccc} 4 & 6 & 8 & 10 & 12 \\ 6 & 9 & 12 & 15 & 18 \\ 8 & 12 & 16 & 20 & 24 \\ 10 & 15 & 20 & 25 & 30 \\ 12 & 18 & 24 & 30 & 36 \end{array}$

T∊	Use the “set membership” operator to find which elements of T are in the table.	0 0 1 0 1
~	Negate the result to identify which elements of T are <u>not</u> in the table. These are the integers which do not have any multiples in the table.	1 1 0 1 0
( )/T	Select the elements of T which we have identified. These are all the primes less than R.	2 3 5

<u>**Equivalent C implementation.**</u>

- Generate a vector of numbers from 1 to R.

```
int arr[6] = {1,2,3,4,5,6};
```

- Drop the first element of the vector and assign the rest to the temporary vector T

```
int temp[5];
for (int i=1;i<=6;i++)
    temp[i-1]=arr[i];
```

- Create the multiplication outer product: a table that holds the result of multiplying each element of T by each element of T

```
int table[5][5];
for(i=0;i<=5;i++)
    for(j=0;j<=5;j++)
        table[i][j] = temp[i]*temp[j];
```

- Use the “set membership” operator to find which elements of T are in the table. Negate the result to identify which elements of T are <u>not</u> in the table. These are all the primes less than R.

```
/* Set membership Algorithm */
/* Negate the binary values and report the respective elements.*/
```

Evaluate the language like APL based on this feature of providing rich operators and high orthogonality?

4. C and C++ use pointers, whereas JAVA and Python don't directly allow the usage of pointers. Evaluate C/C++ based on this feature.

5. Assembly language has few building blocks and there is limited number of ways that they can be combined. Still we find it very hard to read and write the assembly language. What feature according to you is lacking in Assembly language?

6. a. Does the application have any impact on the language we choose for programming?
b. Evaluate C language and MATLAB for programming Operating systems and embedded systems
c. Evaluate the C language for doing matrix multiplication with MATLAB

C Language program

```
include <stdio.h>

int main()
{
  int m, n, p, q, c, d, k, sum = 0;
  int first[10][10], second[10][10], multiply[10][10];

  printf("Enter the number of rows and columns of first matrix\n");
  scanf("%d%d", &m, &n);
  printf("Enter the elements of first matrix\n");

  for (  c = 0 ; c < m ; c++ )
    for ( d = 0 ; d < n ; d++ )
      scanf("%d", &first[c][d]);

  printf("Enter the number of rows and columns of second matrix\n");
  scanf("%d%d", &p, &q);

  if ( n != p )
    printf("Matrices with entered orders can't be multiplied with each other.\n");
  else
  {
    printf("Enter the elements of second matrix\n");

    for ( c = 0 ; c < p ; c++ )
      for ( d = 0 ; d < q ; d++ )
        scanf("%d", &second[c][d]);

    for ( c = 0 ; c < m ; c++ )
    {
```

```
    for ( d = 0 ; d < q ; d++ )
    {
      for ( k = 0 ; k < p ; k++ )
      {
        sum = sum + first[c][k]*second[k][d];
      }

      multiply[c][d] = sum;
      sum = 0;
    }
  }

  printf("Product of entered matrices:-\n");

  for ( c = 0 ; c < m ; c++ )
  {
    for ( d = 0 ; d < q ; d++ )
      printf("%d\t", multiply[c][d]);

    printf("\n");
  }
 }

 return 0;
}
```

MATLAB Program
A = [1 1 0 0]; B = [1; 2; 3; 4]; **Multiply** A times B

7. Use the following grammar and Left most derivation to derive the A = B + C * A

```
<assign> ->  <id> = <expr>

<id>  ->    A | B | C

<expr> -> < expr > + <expr>

              | < expr > * <expr>

               | ( <expr> )

               | <id>
```

Use the right most derivation to derive the same. Check if the parse trees are the same. Why is the compiler complaining about this?

8. How will you remove the ambituity in the above grammar? Take necessary steps for the same and update the grammar.