First Year First Semester Course

M.Tech. (CS) [Batch 2021-23]

Lecture #18

Introduction to Programming

Functional Programming

Programming: Some Guidelines and Efficiency Issues

Course Instructor:

Dr. Monidipa Das

DST-INSPIRE Faculty

Machine Intelligence Unit (MIU), Centre for Artificial Intelligence and Machine Learning (CAIML)

Indian Statistical Institute (ISI) Kolkata, India

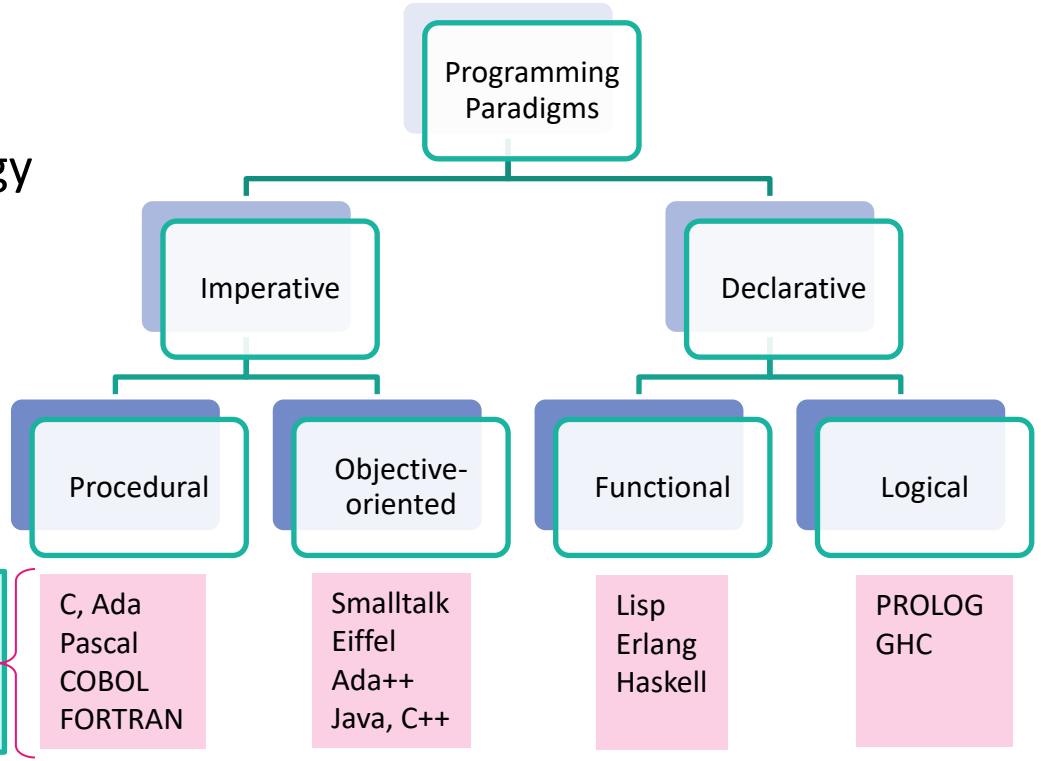
Programming Paradigms

- **Programming paradigm:**

  A mode of thinking aka a programming methodology

- **Programming language:**

  A tool to solve problem using computer



Functional Programming

- Use conditional expressions and recursion to perform computation.

- **Pure Functional Programming Languages**

  – Support only the functional paradigms.

  Example – Haskell

- **Impure Functional Programming Languages**

  – Support the functional paradigms and imperative style programming.

  Example – LISP

LISP

- LISP: derived from "**LISt Processor**"

- Function calls are written with the function name INSIDE the brackets
  - e.g. if our C function call looked like func(x, b, c)

  then the equivalent lisp call would look like (func x b c)

- Everything is done using function calls, even things like variable declarations and assignment operations,
  - e.g. if in C we wanted to declare a variable like int x = 0;

  the equivalent lisp would be (defvar x 0)

- Compound expressions are done using nested function calls,
  - e.g. if in C we wanted to compute x = (3 * y) + (z / 7)

  then the equivalent lisp would be (setf x (+ (* 3 y) (/ z 7)))

LISP: Code Example


```
(defun factorial (N)
"Compute the factorial of N."
(if (= N 1)
  1
(* N (factorial (- N 1)))))
```

```
(defun fibonacci (N)
"Compute the N'th Fibonacci number."
(if (or (zerop N) (= N 1))
   1
(+ (fibonacci (- N 1)) (fibonacci (- N 2)))))
```

```
(defun list-append (L1 L2)
"Append L1 by L2."
(if (null L1)
  L2
(cons (first L1) (list-append (rest L1) L2))))
```

Programming: Some Guidelines

Programming Guidelines

1. Use indentation while writing a program
2. Place the constants on the left of relational operators and variables on the right.

```
int var = 0;
if(0 == var) // Not 'var == 0'
   printf("If statement");
else
   printf("Else statement");
```

3. Avoid equality checking with floating point type variables

```
float f = 0.6;
if(0.6 == f) printf ("If statement");
else printf("Else statement");
```

Output: **Else statement**

Programming Guidelines

4. Instead of assigning values to a large 2-D array defined within a program, we can read the values from an external file.

Example:

```
double MATRIX[SIZE][SIZE] = {
#include "VALUES.txt"
};
```

Note: The preprocessor directive must start in a separate line.

Programming Guidelines

5. Avoiding division

In standard processors, divisions are time-consuming because they take a constant time plus a time for each bit to divide.

```
if((a / b) > c)
  printf("More");
```

It can be efficiently written as follows:

```
if(a > (b * c))
  printf("More");
```

Here, the only assumptions are b is non-negative and b $*$ c fits into an integer. The latter one is also safe if b = 0.

Programming Guidelines

6. Use the right (>>) and left (<<) shift operations instead of integer multiplication and division, wherever respectively possible.
   - Bit level operations are much faster.

7. Try to avoid arithmetic multiplication as and when possible.

```
int m;
printf("%d", m * 17);
```

It should be written using bitwise left shift as follows.

```
int m;
printf("%d", (m << 4) + m);
```

- **Note:** m * n will return the same result as that of (m << p) + m, where n can be represented as $2^p + 1$.

Programming Guidelines

8. Test whether a number is a power of 2 with bitwise AND

```
int m;
scanf("%d", &m);
if((m & (m - 1)) == 0)
  printf("%d is a power of 2", m);
else
  printf("%d is not a power of 2", m)
```

9. Test whether a pair of integers have the same sign with bitwise XOR as follows:

```
int m, n;
scanf("%d%d", &m, &n);
if((m ^ n) < 0)
  printf("%d and %d have different signs", m, n);
else
  printf("%d and %d have the same sign", m, n);
```

Programming Guidelines

10. Never perform modular division with a power of 2.

```
int m;
printf("%d", m % 8);
```

It should be written using bitwise and as follows.

```
int m;
printf("%d", m & 7);
```

Note: m % n will return the same result as that of m & (n-1), where n is a power of 2.

Programming Guidelines

11. Instead of repeatedly dividing by x, compute 1/x and multiply accordingly. It is really beneficial if you do more than 3 divides.

Example:

```
int i, n = 10;
float x = 0.5, result = 1.0;
for(i = 0; i < n; i++)
   result /= x;
```

This can be efficiently done in the following alternative way:

```
int i, n = 10;
float x = 0.5, result = 1.0;
x = result / x;
for(i = 0; i < n; i++)
        result = result * x;
```

Programming Guidelines

12. Avoid the use of pow() for computing small integer powers.

```
int m;
printf("%d", (int)pow(m, 3.0));
```

It should be written as follows.

```
int m;
printf("%d", m * m * m);
```

13. Avoid type casting wherever possible. Integer and floating point instructions often operate on different registers, so a casting requires a copy.

Programming Guidelines

14. Avoid dynamic memory allocation during computation.
    - Allocating memory on the heap is more expensive than adding it on the stack. The operating system needs to perform some computation to find a memory block of the requisite size.

15. Move loops inside function calls.

Replace the following code

```
for(i=0;i<n;i++){
    Function();
}
```

with this code

```
Function(){
    for(i=0;i<n;i++){
    ...}
}
```

Programming Guidelines

16. Use inline functions for replacing short functions to eliminate the function overhead. Hence, the following function

```
int Function(x, y)
{
   x = x - y;
   y++;
   x = x * y;
   return x;
}
```

can be effectively written as follows:

```
#define Function(x, y) (((x)-(y)) * ((y)+1))
```

Optimized Code Example

- Write an optimized C program that takes an integer n from stdin and prints the first n elements (separated by comma) of Fibonacci series on stdout.

Standard recursive implementation:

```
int i = 0;
for(i = 0 ; i <n ; i++){
        printf("%d, ", Fibonacci_Recur(i));
}

int Fibonacci_Recur(int n){ // n is user input
        if(n <= 1)
                return n;
        if(n >= 2)
                return Fibonacci_Recur(n-1) + Fibonacci_Recur(n-2);
}
```

Optimized Code Example [contd.]

Standard iterative implementation:

```
int i, n1 = 0, n2 = 1, n3, n=8; // n is user input
printf("%d, %d, ", n1, n2);
for(i = 0 ; i <n-2 ; i++){
        n3 = n1 + n2;
        printf("%d, ", n3);
        n1 = n2;
        n2 = n3;
}
```

An optimized implementation:

```
int n1 = 0, n2 = 1, n=8;
for(; n >> 1 ; n -= 2){
        printf("%d, %d, ", n1, n2);
        n1 = n1 + n2;
        n2 = n1 + n2;
}
if(n & 1)
        printf("%d", n1);
```

Programming Guidelines

- Use library functions whenever feasible.
- Avoid too many temporary variables.
- Parenthesize to avoid ambiguity.
- Avoid unnecessary branches.
- Choose a data representation that makes the program simple.
- Modularize using procedures and functions.
- Completely avoid the use of goto.
- Make sure all variables are initialized before use.
- Make it right before you make it faster.

Programming Guidelines

- Use the most appropriate data type for variables, as it reduces code and data size and increases performance considerably.
- Use registers for keeping frequently-used variables.
- Global variables are never allocated to registers. So, we should not use them inside critical loops.
- If possible, pass structures by reference (using a pointer to the structure), otherwise the whole thing will be copied onto the stack and passed, which will slow things down.
- Reading chunk of characters at a time from a file is faster that reading character by character.

Programming Guidelines

- Prefer iteration over recursion.
- Long if...else if...else if... chains require lots of jumps for cases near the end of the chain (in addition to testing each condition). If possible, convert to a switch statement, which the compiler sometimes optimizes into a table lookup with a single jump. If a switch statement is not possible, put the most common clauses at the beginning of the if chain.
- If you do not need a return value from a function, do not define one.

Do it now

- Write a C program that reads a .txt file and replaces a particular word with another word, given by the user.

- Write a C program that computes the determinant of a matrix

Questions?