WEEK-7A

Q:Write C/C++/Java/Python program for classifying the various types of coupling.

AIM:- write C-program for classifying and demonstrating various types of coupling

Theory:- coupling is a measure of the functional strength between two modules or it is a measure of the degree of interaction (or interdependence) between the two modules.
The degree of coupling between two modules depends on their ***interface complexity*****.**
The interface complexity is basically determined ***by the number of types of parameters that are interchanged*** while invoking the functions of the module.
Module coupling can be **Tightly or Loosely** coupled based on the dependencies

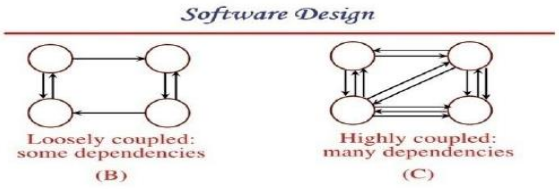

Fig: Module Coupling

Classification of the different types of coupling will help to quantitatively estimate the degree of coupling between two modules.
There are Six types of coupling can occur between any two modules.
This is shown in the figure given below:

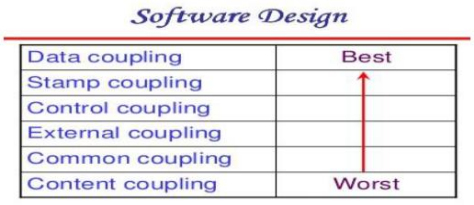

Fig: Types of Coupling

i) **Data coupling:** Two modules are data coupled, if they communicate through a parameter. An example is an elementary data item passed as a parameter between two modules, e.g. an integer, a float, a character, etc. This data item should be problem related and not used for the control purpose.

ii) **Stamp coupling:** Two modules are stamp coupled, if they communicate using a composite data item such as a record /array/structures.

iii) **Control coupling:** Control coupling exists between two modules, if data from one module is used to direct the order of instructions execution in another. An example of control coupling is a flag set in one module and tested in another module.

iv) **External Coupling:** It occurs when two modules share an externally imposed data format, communication protocol, or device interface. All the modules share the same I/O device or the external environment.

v) **Common coupling:** Two modules are common coupled, if they share data through some global data items.

vi) **Content coupling:** Content coupling exists between two modules, if they share code, e.g. a branch from one module into another module.

C-programs for classifying and demonstrating various types of coupling.

1. DATA COUPLING

```
#include<stdio.h>
void datacoupling();
int sum(int , int);
main()
{
datacoupling();
printf("\n DATA COPLING successfully DEMONSTRATED ");
}
//MODULE-A
void datacoupling()
{
int x=10, y=20,k;
k=sum(x,y);
printf("\n the Function SUM return %d",k);
}

// MODULE-B
int sum (int p, int q)
{
    int r=p+q;
    return r;
}
```

Results:-

2. **STAMP coupling**

```
#include<stdio.h>
void stampcoupling();
int total (struct DATA p);

struct DATA
 {
        int x,y;
};

main()
{
stampcoupling();
printf("\n STAMP COPLING successfully DEMONSTRATED ");
}
//MODULE-A
void stampcoupling()
{
int k;
struct DATA q={2,3};
k=total(q);
printf("\n the Function SUM return %d",k);
}

//MODULE-B
int total(struct DATA d )
{
        int r;
        r=d.x+d.y;
        return r;
}
```

Results:

3.COMMON COUPLING

```
#include <stdio.h>
// Global variable
int d = 0;
// Function prototypes
void processA();
void processB();

int main() {
   // Control coupling: main function controls the flow of execution
   processA();
   processB();
   return 0;
}

// MODULE-A
//Function definition for processA
void processA()
{
   // Accessing shared data
   d = 10;
   printf("Process A: Shared data set to %d\n", d);
}

// MODULE-B
//Function definition for processB
void processB() {
   // Accessing shared data
     printf("Process B: Shared data accessed with value %d\n", d);
}
```

4.Controll COPLING

```
#include <stdio.h>
void processA(int value);
void processB(int value);

int main()
{
   int data = 10;
   if (data < 20)
{
  processA(data);
   }
else
 {
      processB(data);
   }
printf("\n Control coupling successfully Demonstrated");
   return 0;
}
// Module
void processA( value)
 {
   printf("Processing in A with value: %d\n", value);
}


// Module
void processB(value)
{
   printf("Processing in B with value: %d\n", value);
}
```

RESULT:-

5.CONTENT COUPLING

```
#include <stdio.h>
void processA(int ) ;
void processB(int ) ;
void processData(int , void (*process)(int)) ;

int main()
{
  int data = 10;
  // Content coupling: Main function knows about the internal details of processA and processB
  processData(data, processA);
  processData(data, processB);

  return 0;
}

// Module A
void processA(int value)
{
  printf("Process A: Received value %d\n", value);
}

// Module B
void processB(int value)
{
  printf("Process B: Received value %d\n", value);
}

// Module C
void processData(int value, void (*process)(int))
{
  // Content coupling: Module C knows the internal implementation of processA and processB
  process(value);
}
```

Result:-