# 1.Calculating Memory Usage

- **Description**

  A set of functions used to calculate the memory usage of a program, i.e. how much memory is consumed by each of the program segements. To get this data from the OS procfs is used.

- **Input**

  A C++ Program. To this we add the relevant functions created for calculting Memory Usage.

- **Output**

  Memory space consumed by :

  i. Data Segment
  ii. Stack Segment
  iii. Text Segment
  iv. Shared Libraries
  v. Heap
  vi. Total Memory Used

- **Usage**

```
g++ <inputFile>.cpp -o MemoryUsage
./MemoryUsage
```

- **Example**

  - **Input**

```
#include<iostream>
#include<fstream>
#include<sstream>

using namespace std;

int getMemoryUsage();
int getMemoryUsageHeap();

int main()
{
    int a[21024];
    int total=0;
    total=getMemoryUsage()+getMemoryUsageHeap();
    cout<<"Total Memory Used  : "<<total<<"KB.\n";
}

int getMemoryUsage()
{
    unsigned dataSegment=0,stackSegment=0,textSegment=0,sharedLib=0;

    ifstream in("/proc/self/status");
    while(in)
    {
        string line,tag;
        getline(in,line);
        istringstream iss(line);
        iss>>tag;
```

```
        if(tag=="VmData:")
        {
            iss>>dataSegment;
        }
        else if(tag=="VmStk:")
        {
            iss>>stackSegment;
        }
        else if(tag=="VmExe:")
        {
            iss>>textSegment;
        }
        else if(tag=="VmLib:")
        {
            iss>>sharedLib;
            break;
        }

    }
    in.close();
    cerr<<"Memory used by Data Segment :  "<<dataSegment<<"KB.\n";
    cerr<<"Memory used by Stack Segment : "<<stackSegment<<"KB.\n";
    cerr<<"Memory used by Text Segment : "<<textSegment<<"KB.\n";
    cerr<<"Memory used by Shared Libraries : "<<sharedLib<<"KB.\n";
    unsigned total=dataSegment+stackSegment+textSegment+sharedLib;
    return total;
}

int getMemoryUsageHeap()
{
    unsigned x=0;
    ifstream in("/proc/self/smaps");
    while(in)
    {
        string line,tag,temp;
        getline(in,line);
        istringstream iss(line);
        for(int i=0;i<6;i++)
        {
            iss>>tag;
        }
        if(tag=="[heap]")
        {
            getline(in,line);
            istringstream iss(line);
            iss>>temp;
            iss>>x;
            break;
        }
    }
    in.close();
    cerr<<"Memory used by Heap = "<<x<<"KB.\n";
    return x;
}
```

- **Output:**

```
Memory used by Data Segment :  192KB.
Memory used by Stack Segment : 96KB.
Memory used by Text Segment : 8KB.
Memory used by Shared Libraries : 2684KB.
Memory used by Heap = 132KB.
Total Memory Used  : 3112KB.
```

# 2.Comparing Executables

- **Description**

It is used to determine a confidence value for whether two programs compute the same function or not. To determine the confidence value we compare the output of the two executables for a fixed number of inputs(,say n). Even if one mismatch is obtained in between the n comparisions, we stop the comparision there and declare them as programs not computing the same function. Else, we return a confidence value at the end which is calculted as follows :

```
Confidence Value = 1 - 1/2^n
```

- **Input**

  Two C++ executables to be compared, number of test cases

- **Output**

  Confidence Value

- **Usage**

```
g++ CompareExeuctables.cpp
./a.out <Exec1Name> <Exec2Name> <nTestCases>
```

- **Examples**
  - **Successful Case**
    - **Code for Executable 1**

```
#include <iostream>
#include <stdlib.h> //for atoi
using namespace std;

int main(int argc, char* argv[])
{
    int temp = 3*atoi(argv[1]);
    cout<<temp;
}
```

    - **Code for Executable 2**

```
#include <iostream>
#include <stdlib.h> //for atoi
using namespace std;

int main(int argc, char* argv[])
{
    int temp = 3*atoi(argv[1]);
    cout<<temp;
}
```

    - **Output**

```
Passed for all 10 test cases.
Confidence Value calculated as 1 - (1/2^n) : 0.999023
```

  - **Unsuccessful case**
    - **Code for Executable 1**

```
#include <iostream>
#include <stdlib.h> //for atoi
using namespace std;

int main(int argc, char* argv[])
{
    int temp = 3*atoi(argv[1]);
    cout<<temp;
```

```
}
```

- **Code for Executable 2**

```
#include <iostream>
#include <stdlib.h> //for atoi
using namespace std;

int main(int argc, char* argv[])
{
    int temp = 2*atoi(argv[1]);
    cout<<temp;
}
```

- **Output**

```
Failed for test case i= 1.
```

# 3.Comparing Control Flow Graph Structures

- **Description**

  Here the structure of the Control Flow Graph(CFG) is compared to determine the similarity between two programs. The CFG is extracted from the C programs using the **-fdump-tree-cfg** option which gives a **.cfg** file as output.

  - **An example for a simple program**

    - **Code**

```
#include<stdio.h>
void main()
{
    int i,j,k=0;
    for(i=0;i<3;i++)
    {
        for(j=0;j<=3;j++)
        {
            k = 5;
            printf("%d %d",i,j);
        }
    }
}
```

    - **CFG Generation**

```
gcc -fdump-tree-cfg example.c -o example
```

    - **CFG Generated**

```
;; Function main (main)

main ()
{
  int k;
  int j;
  int i;
  const char * restrict D.1710;

<bb 2>:
  k = 0;
  i = 0;
  goto <bb 7>;

<bb 3>:
```

```
  j = 0;
  goto <bb 5>;

<bb 4>:
  k = 5;
  D.1710 = (const char * restrict) "%d %d";
  printf (D.1710, i, j);
  j = j + 1;

<bb 5>:
  if (j <= 3)
    goto <bb 4>;
  else
    goto <bb 6>;

<bb 6>:
  i = i + 1;

<bb 7>:
  if (i <= 2)
    goto <bb 3>;
  else
    goto <bb 8>;

<bb 8>:
  return;

}
```

A parser is then used along with a lexical analyzer to extract the structure from the .cfg file. The lexical analyzer is made using flex and the parser using bison.

Lexical Analyzer Code
Parser Code

- **Structure Extracted from above Example**

```
<bb 2>:
goto <bb 7>

<bb 3>:
goto <bb 5>

<bb 4>:

<bb 5>:
goto <bb 4>
goto <bb 6>

<bb 6>:

<bb 7>:
goto <bb 3>
goto <bb 8>

<bb 8>:
```

To compare the two structures we use a script using diff which returns the number of lines in each file and difference count.

Difference Script Code

- **Input**

  Two text files containing the structure of CFG.

- **Output**

  Difference count, number of lines in each file.

- **Usage**

```
./diffCount.sh <file1> <file>
```

- **Example**

  - **Input**

    - **Structure for InsertionSort1**

```
<bb 2>:
goto <bb 4>

<bb 3>:

<bb 4>:
goto <bb 3>
goto <bb 5>

<bb 5>:
goto <bb 11>

<bb 6>:
goto <bb 8>

<bb 7>:

<bb 8>:
goto <bb 9>
goto <bb 10>

<bb 9>:
goto <bb 7>
goto <bb 10>

<bb 10>:

<bb 11>:
goto <bb 6>
goto <bb 12>

<bb 12>:
goto <bb 14>

<bb 13>:

<bb 14>:
goto <bb 13>
goto <bb 15>

<bb 15>:
```

    - **Structure for InsertionSort2**

```
<bb 2>:
goto <bb 4>

<bb 3>:

<bb 4>:
goto <bb 3>
goto <bb 5>
```

```
<bb 5>:
goto <bb 11>

<bb 6>:
goto <bb 8>

<bb 7>:

<bb 8>:
goto <bb 9>
goto <bb 10>

<bb 9>:
goto <bb 7>
goto <bb 10>

<bb 10>:

<bb 11>:
goto <bb 6>
goto <bb 12>

<bb 12>:
goto <bb 14>

<bb 13>:

<bb 14>:
goto <bb 13>
goto <bb 15>

<bb 15>:
```

- **Output**

```
No. of lines in file1 : 42
No. of lines in file2 : 42
Difference Count : 0
```

A complete list of test cases and their results can be found here.