CS 1203: Data Structures **Semester:** Monsoon 2023

Assignment #2

Instructor: Subhamoy Mitra
Due: September 15, 2023

Student Name: Varsha Baisane
varsha.baisane_ug25#@ashoka.edu.in

Question 3

The maximum dimension of a C program address which is also the maximum addressable memory, depends upon the architecture and operating systems on which the program is running.

Like for a 32-bit system, the maximum addressable memory is typically limited to 4 gigabytes ($2^{32}bytes$). This means that a C program running on a 32-bit system can address up to 4 GB of memory.

64-bit Systems: For a 64-bit systems, the maximum addressable memory space is vast, theoretically allowing access to 18-Quintillion bytes ($2^{64}bytes$) of memory. In other words, any amount of memory greater than 4 GB can be easily handled by it.

Operating systems and compiler:The effective maximum addressable memory space can also be influenced by the operating system and compiler settings. For security and resource management reasons, several operating systems limit the amount of memory that a single process can consume.

In summary, the maximum address space dimension of a C program is determined by several factors, including the system architecture (32-bit or 64-bit), the individual hardware, the operating system, and system settings.

```c
#include <stdio.h>

int main() {
    unsigned int n = 4294967294, i, prod = 1;

    for (i = 2; i <= n; i++)
        prod = prod * i;

    printf("%d\n", prod);
}
```

```c
#include <stdio.h>

int square(int);

int main() {
    int x = 7;
    printf("x: %d, square(x): %d\n", x, square(x));
    printf("Address of the function main is: %p\n", (void*)main);
    printf("Address of the function square is: %p\n", square);
}

int square(int x) {
    return x*x;
}
```

```c
#include <stdio.h>

void decToBinary(int n) {
    int binaryNum[32];
    int zeros[32];
    int ones[32];

    int i = 0;
    int zeroCount = 0;
    int oneCount = 0;

    while (n > 0) {
        binaryNum[i] = n % 2;
        if (binaryNum[i] == 0) {
            zeros[zeroCount] = binaryNum[i];
            zeroCount++;
        } else {
            ones[oneCount] = binaryNum[i];
            oneCount++;
        }
        n = n / 2;
        i++;
    }

    // printing binary array in reverse order
    printf("Binary representation: ");
    for (int j = i - 1; j >= 0; j--)
        printf("%d", binaryNum[j]);
    printf("\n");

    // printing zeros and ones
    printf("Zeros: ");
    for (int j = 0; j < zeroCount; j++)
        printf("%d", zeros[j]);
    printf("\n");

    printf("Ones: ");
    for (int j = 0; j < oneCount; j++)
        printf("%d", ones[j]);
    printf("\n");
}

int main() {
    int n = 12;
    decToBinary(n);
    return 0;
```

```
}
```