Program 1

```c
#include <stdio.h>
#include <stdlib.h>

int main() {

// Deklarasi dan alokasi array dinamis
int *dynamicArray = (int *)malloc(5 * sizeof(int));

// Inisialisasi elemen array dinamis
for (int i = 0; i < 5; ++i) {
    dynamicArray[i] = i + 1;
}
// Akses dan cetak elemen array
for (int i = 0; i < 5; ++i) {
    printf("%d ", dynamicArray[i]);
}

// Dealokasi array dinamis
free(dynamicArray);
return 0;
}
```

1. Tuliskan output program di atas!
   1 2 3 4 5
2. Apakah yang terjadi jika baris 13 dihapus?
   Program setelah berjalan memory masih ada di sistem os sehingga ruang memory tidak kembali, free() digunakan untuk membebaskan memory yang sudah dialokasikan.

Program 2

```c
#include <stdio.h>
#include <stdlib.h>

int main() {

// Deklarasi dan alokasi array dinamis
int *dynamicArray = (int *)malloc(5 * sizeof(int));

// Inisialisasi elemen array dinamis
for (int i = 0; i < 5; ++i) {
    dynamicArray[i] = i + 1;
}
// Akses dan cetak elemen array
for (int i = 0; i < 5; ++i) {
```

```
    printf("%d ", dynamicArray[i]);
}

int newSize = 8;
dynamicArray = (int *)realloc(dynamicArray, newSize * sizeof(int));

for (int i = 5; i < newSize; ++i) {
    dynamicArray[i] = i + 1;
}
for (int i = 0; i < newSize; ++i) {
    printf("%d ", dynamicArray[i]);
}

// Dealokasi array dinamis
free(dynamicArray);
return 0;
}
```

3. Tuliskan output program di atas! Apakah isi array yang sudah ada sebelumnya berubah?
   Output: 1 2 3 4 5 1 2 3 4 5 7 8
   Isi array data sebelumnya tidak berubah karena penambahan isi array yang baru dimulai dari iterasi 5 dari array sebelumnya sehingga hanya menambahkan data baru “6 7 8” di akhir array. Output berulang 1 2 3 4 5 karena terprint di sebelum data diubah.

4. Apa yang dapat Anda simpulkan mengenai penggunaan malloc dan realloc?
   Malloc digunakan untuk mengalokasikan data (seperti memesan lokasi memory) supaya dapat diisi suatu data. Sedangkan untuk realloc digunakan untuk mengubah data yang sudah dialokasikan.

Program 3

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int n, i;
    int *a, *b;

    scanf("%d", &n);
    a = (int*) malloc(n * sizeof(int));
    b = (int*) malloc(n * sizeof(int));

    for (i = 0; i < n; i++) {
        a[i] = rand() % 10;
        b[n-i-1] = *(a + i);
    }
```

```
    for (i = 0; i < n; i++) printf("%d ", a[i]);
    for (i = 0; i < n; i++) printf("%d ", b[i]);

    free(a);
    free(b);

    return 0;
}
```

5. Ubahlah bagian berwarna kuning menggunakan pointer style untuk mengakses array. Simpan perubahan yang Anda lakukan dalam Prog3_NIM. cpp.

Prog3_G6401231038.cpp

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int n, i;
    int *a, *b;

    scanf("%d", &n);
    a = (int*) malloc(n * sizeof(int));
    b = (int*) malloc(n * sizeof(int));

    for (i = 0; i < n; i++) {
        *(a + i) = rand() % 10;
        *(b + n - i - 1) = *(a + i);
    }

    for (i = 0; i < n; i++) printf("%d ", *(a + i));
    for (i = 0; i < n; i++) printf("%d ", *(b + i));

    free(a);
    free(b);

    return 0;
}
```

Program 4

```
#include <bits/stdc++.h>
using namespace std;
```

```
int* getValues()
{
    int *arr = (int *) malloc(10*sizeof(int)); for (int i = 0; i < 10; i++)
    arr[i] = i + 1;

    return arr;
}

int main()
{

    int* array;
    array = getValues();

    for (int i = 0; i < 10; i++) {
        cout << "*(array + " << i << ") : ";
        cout << *(array + i) << endl;
    }

    return 0;
}
```

Program 5

```
#include <bits/stdc++.h>
using namespace std;

vector<int> getValues()
{
    vector<int> v;
    for (int i = 0; i < 10; i++) v.push_back(i + 1);
    return v;
}

int main()
{
    vector<int> get;
    get = getValues();

    vector<int>::iterator it;
    for (it = get.begin(); it != get.end(); ++it)
        cout << *it << " ";

return 0;
}
```

6. Jalankan PROGRAM 4 dan PROGRAM 5 di atas! Apakah outputnya sama?
   Berbeda format, akan tetapi masih menghasilkan nilai yang sama.
   Output Program 4:

```
*(array + 0) : 1
*(array + 1) : 2
*(array + 2) : 3
*(array + 3) : 4
*(array + 4) : 5
*(array + 5) : 6
*(array + 6) : 7
*(array + 7) : 8
*(array + 8) : 9
*(array + 9) : 10
```

   Output Program 5:

```
1 2 3 4 5 6 7 8 9 10
```

7. Apakah it, begin() dan end() pada PROGRAM 5?
   it: Sebagai iterator untuk mengakses isi vektor
   begin(): awal dari isi vektor, yang di sini vektor awalnya dari vektor get yang bernilai 1.
   end(): akhir dari isi vektor, yang di sini vektor awalnya dari vektor get yang bernilai 10.

8. Simpulkan perbedaan array dan vector dari kedua program di atas!
   Program keduanya memiliki keuntungan dan kekurangan masing-masing dengan masing-masing memiliki caranya sendiri dalam penggunaannya.
   Pada penggunaan ***array*** dilakukan dengan alokasi manual dengan ukuran yang tetap, array dapat diakses dengan menggunakan pointer/indeks. Akan tetapi penggunaan array dapat beresiko memori leak.
   Pada penggunaan ***vektor*** untuk pengalokasian ukuran sebagaimana ukuran data (dinamis) dan untuk manajemen memory dilakukan otomatis oleh compiler.

9. Pada PROGRAM 5, tambahkan beberapa baris perintah untuk melakukan penghapusan seluruh elemen sebelum return 0, menggunakan erase() atau pop_back(). Simpan hasilnya pada PROGRAM 5 (Prog5_NIM.cpp) .

```cpp
#include <bits/stdc++.h>
using namespace std;

vector<int> getValues()
{
    vector<int> v;
    for (int i = 0; i < 10; i++) v.push_back(i + 1);
    return v;
}

int main()
{
    vector<int> get;
    get = getValues();
```

```
    get.erase(get.begin());
    get.pop_back();

    vector<int>::iterator it;
    for (it = get.begin(); it != get.end(); ++it)
        cout << *it << " ";

return 0;
}
```

Output:

```
2 3 4 5 6 7 8 9
```

Program 6

```
#include <bits/stdc++.h>
using namespace std;

void search(char* pat, char* txt)
{
        int M = strlen(pat);
        int N = strlen(txt);

        for (int i = 0; i <= N - M; i++) {
                int j;
                for (j = 0; j < M; j++)
                        if (....)
                        break;

                if (....)
                        cout << "Pattern found at index " << i << endl;
                }
}

int main()
{
        char txt[] = "AABAACAADAABAAABAA";
        char pat[] = "AABA";
        search(pat, txt);
        return 0;
}
```

10. Jalankan program di atas dengan melengkapi bagian kotak biru! Simpan hasilnya pada PROGRAM 6 (prog6_NIM. cpp). Apa outputnya?

```
#include <bits/stdc++.h>
using namespace std;

void search(char* pat, char* txt)
{
        int M = strlen(pat);
        int N = strlen(txt);

        for (int i = 0; i <= N - M; i++) {
                int j;
                for (j = 0; j < M; j++)
                        if (txt[i+j]!= pat[j])
                        break;

                if (j==M)
                        cout << "Pattern found at index " << i << endl;
                }
}

int main()
{
        char txt[] = "AABAACAADAABAAABAA";
        char pat[] = "AABA";
        search(pat, txt);
        return 0;
}
```

Output:

```
Pattern found at index 0
Pattern found at index 9
Pattern found at index 13
```

11. Jelaskan langkah-langkah algoritme yang digunakan pada PROGRAM 6 menggunakan ilustrasi yang sesuai. Bagaimanakah menghitung kompleksitasnya?
    Program membuat fungsi search yang digunakan untuk mencari patern dari pat[] pada txt[], dengan pertama kali yang dilakukan adalah menghitung panjang char dari txt[] dan pat[] kemudian diselisihkan untuk sebagai iterasinya (supaya tidak melakukan iterasi yang tidak perlu). Kemudian pada iterasi j melakukan penyamaan char yang dimana jika char pada txt[] di index tertentu sama pada pat[] di index tertentu, maka program akan looping untuk menyamakan kesuluruhan patern dengan membandingkan panjang dari potongan txt tersebut, apabila tidak ditemukan program akan menskip huruf itu atau bila sampai tidak ada maka tidak menghasilkan output. Dengan algoritma tersebut maka fungsi search dapat digunakan dan digunakan pada program pada search(pat, txt).

Untuk kompleksitas program sebesar O((N-M+1)*M) karena apabila selisih panjang patern dengan txt tidak jauh maka iterasi semakin sedikit yang kompleksitasnya (N-M+1) dan *M karena jika semakin dekat di awal patern ditemukan maka semakin cepat juga program selesai. Sehingga kompleksitas program sebesar **O((N-M+1)*M)**