Nama : Gendhi Ramona Prastyo
Nim : 1203230011
Prodi : IF 03-02

SOAL 1

Output :

Penjelasan :

1. Pada bagian struct memliki dua 2 bagian yaitu

a) char alphabet yang digunakan untuk memberikan nilai alphabet yang di inisialisasikan pada fungsi main

b) struct node *link memberikan pointer terhada struct yang sama yaitu struct node itu sendiri yang digunakan untuk menyimpan nilai memori address yang di inisialisasikan pada fungsi main

2. Pada fungsi main membuat variabel baru l1-l9 yang di inisialisasikan dengan struct node yang digunakan untuk inisialisasi terhadap *link dan alphabet

3. Setiap variable baru yang dibreikan l?.link = NULL diberikan nilai baru yaitu memori adress dari variabel l1-l9 yang sudah di inisialisasikan di awal tetapi tetap mengikuti alur soal modul dimulai dari N->F->O->M->K->A->T->R->I yang mana setiap l?.link di berikan nilai memori address dengan ketentuan yang ada

4. Proses selanjutnya meng-outputkan masing-masing dari l4.link diikuti dengan link dan juga alphabet setelah mencapai Huruf I l4.link dan l5.link di berikan nilai memori address yang baru agar dapatt mengoutputkan dua huruf terakhir yang di inginkan.

SOAL 2

```
int twoStacks(int maxSum, int a_count, int *a, int b_count, int *b) {
  int i = 0, j = 0, sum = 0, count = 0;
  while (i < a_count && sum + a[i] <= maxSum) {
    sum += a[i];
    i++;
  }
  count = i;
  while (j < b_count && i >= 0) {
    sum += b[j];
    j++;
    while (sum > maxSum && i > 0) {
      i--;
      sum -= a[i];
    }
    if (sum <= maxSum && i + j > count) {
      count = i + j;
    }
  }
  return count;
}
```

Penjelasan :

1. Iterasi melalui stack pertama:

Kode ini mengiterasi elemen-elemen stack pertama (a) selama jumlah elemen (sum) tidak melebihi maxSum.

Di setiap iterasi, kode menambahkan elemen saat ini dari stack pertama ke sum dan meningkatkan penghitung i.

Loop ini melacak berapa banyak elemen yang dapat diambil dari stack pertama tanpa melebihi maxSum.

2. Inisialisasi count:

Variabel count diinisialisasi dengan nilai i saat ini. Ini mewakili jumlah maksimum elemen yang dapat diambil dari stack pertama sejauh ini.

3. Iterasi melalui stack kedua:

Kode kemudian mengiterasi elemen-elemen stack kedua (b).

Di setiap iterasi, kode menambahkan elemen saat ini dari stack kedua ke sum dan meningkatkan penghitung j.

4. Menyesuaikan elemen dari stack pertama (jika perlu):

Di dalam loop kedua, ada loop bersarang yang beriterasi selama sum melebihi maxSum dan ada elemen di stack pertama (i lebih besar dari 0).

Di setiap iterasi loop bersarang, kode ini menghapus elemen terakhir dari stack pertama dan mengurangkan nilainya dari sum. Ini memastikan bahwa sum tidak melebihi maxSum.

5. Perbarui count (opsional):

Setelah menyesuaikan elemen dari stack pertama (jika perlu), kode ini memeriksa apakah sum saat ini kurang dari atau sama dengan maxSum dan jumlah gabungan elemen dari kedua stack (i + j) lebih besar dari count saat ini.

Jika kedua kondisi benar, kode ini memperbarui count ke jumlah elemen gabungan (i + j). Ini memastikan bahwa count mewakili jumlah maksimum elemen yang dapat diambil dari kedua stack tanpa melebihi maxSum.

6. Return count:

Setelah mengiterasi kedua stack, fungsi ini mengembalikan nilai akhir count, yang mewakili jumlah maksimum elemen yang dapat diambil dari kedua stack tanpa melebihi maxSum.