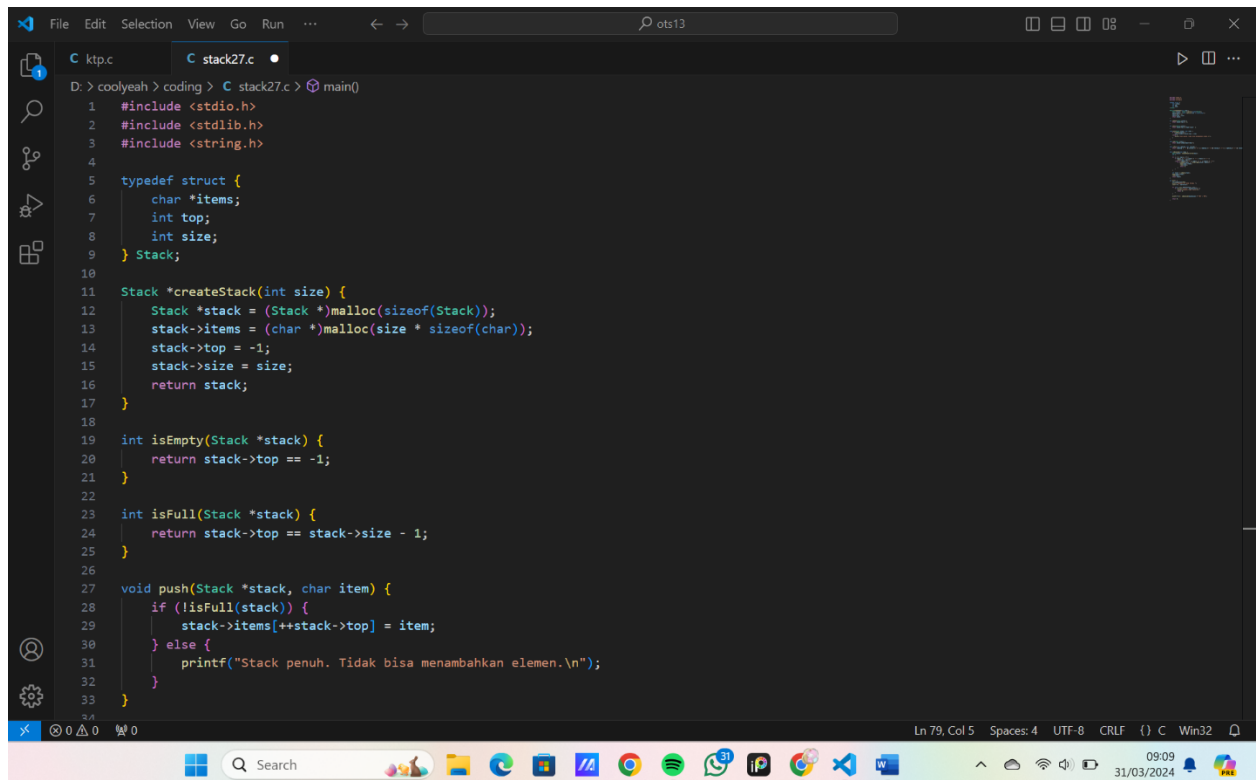


**NAMA : MARIA ROSA WAHYUNING UTAMI**  
**NIM : 1203230123**  
**KELAS : IF 03-03**

## **TUGAS OTS – STACK**

**RABU, 27 MARET 2024**

### **1. Source Code**



```
File Edit Selection View Go Run ...
C ktp.c C stack27.c
D:\> coolyeah > coding > C stack27.c > main()
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 typedef struct {
6     char *items;
7     int top;
8     int size;
9 } Stack;
10
11 Stack *createStack(int size) {
12     Stack *stack = (Stack *)malloc(sizeof(Stack));
13     stack->items = (char *)malloc(size * sizeof(char));
14     stack->top = -1;
15     stack->size = size;
16     return stack;
17 }
18
19 int isEmpty(Stack *stack) {
20     return stack->top == -1;
21 }
22
23 int isFull(Stack *stack) {
24     return stack->top == stack->size - 1;
25 }
26
27 void push(Stack *stack, char item) {
28     if (!isFull(stack)) {
29         stack->items[++stack->top] = item;
30     } else {
31         printf("Stack penuh. Tidak bisa menambahkan elemen.\n");
32     }
33 }
```

Ln 79, Col 5 Spaces: 4 UTF-8 CRLF Win32

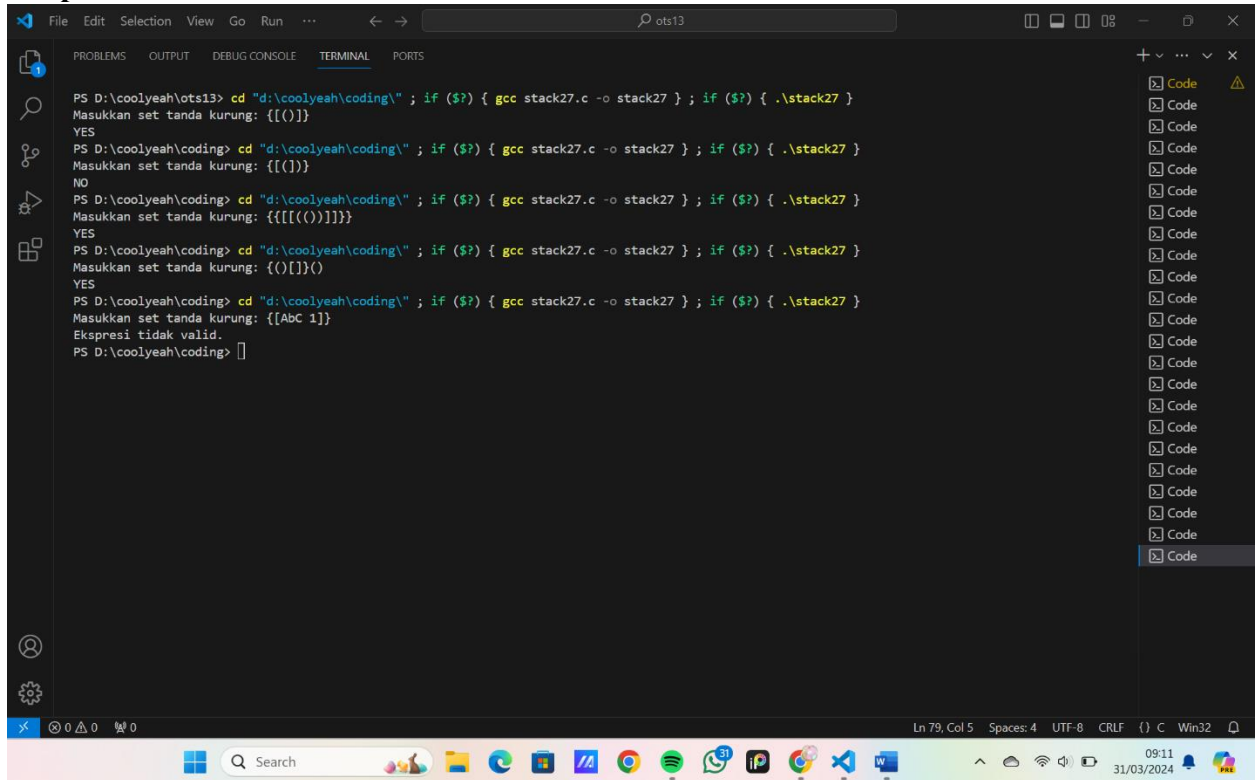
```
File Edit Selection View Go Run ... 0ts13
C ktp.c C stack27.c
D:\> coolyyeah > coding > C stack27.c > main()

34
35 char pop(Stack *stack) {
36     return stack->items[stack->top--];
37 }
38
39 char isPair(char opening, char closing) {
40     return (opening == '(' && closing == ')') || (opening == '{' && closing == '}') || (opening == '[' && closing == ']');
41 }
42
43 char isBalanced(char *exp) {
44     Stack *stack = createStack(strlen(exp));
45     int i;
46
47     for (i = 0; exp[i]; i++) {
48         if (exp[i] == '(' || exp[i] == '{' || exp[i] == '[') {
49             push(stack, exp[i]);
50         } else if (exp[i] == ')' || exp[i] == '}' || exp[i] == ']') {
51             if (isEmpty(stack) || !isPair(pop(stack), exp[i])) {
52                 free(stack->items);
53                 free(stack);
54                 return 0;
55             }
56         }
57     }
58
59     int result = isEmpty(stack);
60     free(stack->items);
61     free(stack);
62     return result;
63 }
64
65 int main() {
66     char expression[100];
```

```
File Edit Selection View Go Run ... 0ts13
C ktp.c C stack27.c
D:\> coolyyeah > coding > C stack27.c > main()

65 int main() {
66     char expression[100];
67     printf("Masukkan set tanda kurung: ");
68     scanf("%s", expression);
69
70     for (int i = 0; expression[i]; i++) {
71         if ( strchr("{}[]", expression[i]) ) {
72             printf("Ekspresi tidak valid.\n");
73             return 0;
74         }
75     }
76
77     printf("%s\n", isBalanced(expression) ? "YES" : "NO");
78
79     return 0;
80 }
```

## 2. Output



```
PS D:\coolyeah\ots13> cd "d:\coolyeah\coding\" ; if ($?) { gcc stack27.c -o stack27 } ; if ($?) { .\stack27 }
Masukkan set tanda kurung: [[()]]
YES
PS D:\coolyeah\coding> cd "d:\coolyeah\coding\" ; if ($?) { gcc stack27.c -o stack27 } ; if ($?) { .\stack27 }
Masukkan set tanda kurung: [[[([)])]]
NO
PS D:\coolyeah\coding> cd "d:\coolyeah\coding\" ; if ($?) { gcc stack27.c -o stack27 } ; if ($?) { .\stack27 }
Masukkan set tanda kurung: ([)]()
YES
PS D:\coolyeah\coding> cd "d:\coolyeah\coding\" ; if ($?) { gcc stack27.c -o stack27 } ; if ($?) { .\stack27 }
Masukkan set tanda kurung: [Abc 1]
Eksresi tidak valid.
PS D:\coolyeah\coding> 
```

## 3. Penjelasan

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

Mendeklarasikan library standar untuk input dan output standar, seperti printf() dan scanf()

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

Mendeklarasikan library yang menyediakan fungsi-fungsi umum seperti alokasi memori dengan malloc() dan free(), konversi string menjadi angka

```
#include <string.h>
```

Mendeklarasikan library standar untuk pengolahan string, seperti mencari panjang string dengan strlen(), menyalin string dengan strcpy(), membandingkan string dengan strcmp(), dan sebagainya

```
typedef struct {
```

Mendefinisikan struktur data baru yang akan digunakan untuk merepresentasikan stack dalam program

```
char *items;
```

Membuat variabel yang menunjuk ke alamat memori dimana elemen-elemen stack disimpan. **char \*** mengindikasikan bahwa setiap elemen tumpukan adalah karakter, dan **items** adalah nama variabel yang menyimpan alamat memori

```
int top;
```

Mendefinisikan variabel **top** yang akan menyimpan indeks dari elemen teratas dalam tumpukan. Variabel ini menunjukkan posisi teratas dari tumpukan, sehingga dapat menambah atau menghapus elemen dari sana

```
int size;
```

Mendefinisikan variabel **size** yang akan menyimpan ukuran kapasitas maksimum dari stack. Variabel ini digunakan untuk membatasi jumlah maksimum elemen yang dapat disimpan dalam stack

```
} Stack;
```

Menutup definisi struktur data **Stack** yang dimulai dengan **typedef struct {**. Serta memberikan nama struktur data yang telah didefinisikan sebelumnya dengan nama **Stack**, sehingga dapat membuat variabel bertipe **Stack** di dalam program dan menggunakan struktur data tersebut untuk merepresentasikan tumpukan

```
Stack *createStack(int size) {
```

Deklarasi fungsi yang bernama **createStack** yang mengambil satu argument berupa integer **size** yang menentukan ukuran tumpukan yang ingin dibuat. Fungsi ini akan mengembalikan pointer ke tumpukan yang baru dibuat

```
Stack *stack = (Stack *)malloc(sizeof(Stack));
```

Melakukan alokasi memori untuk tumpukan menggunakan fungsi **malloc()**. **sizeof(Stack)** memberikan ukuran memori yang diperlukan untuk menyimpan tumpukan, dan **malloc()** mengalokasikan memori sebesar itu. Hasilnya disimpan dalam variabel pointer **stack** yang akan menunjuk ke tumpukan yang baru dibuat

```
stack->items = (char *)malloc(size * sizeof(char));
```

Melakukan alokasi memori untuk array yang menyimpan elemen-elemen tumpukan. **size \* sizeof(char)** mengalokasikan memori sebesar **size** kali ukuran satu karakter. Hasilnya disimpan dalam **items** di dalam **stack**

```
stack->top = -1;
```

Menginisialisasi indeks **top** dari tumpukan menjadi -1, menandakan bahwa tumpukan kosong pada awalnya

```
stack->size = size;
```

Menetapkan ukuran tumpukan yang baru dibuat sesuai dengan nilai yang diberikan saat memanggil fungsi **createStack**

```
return stack;
```

Mengembalikan pointer **stack** yang menunjuk ke tumpukan yang baru dibuat kepada pemanggil fungsi **createStack**. Dengan demikian, pemanggil fungsi dapat menggunakan tumpukan yang baru dibuat untuk melakukan operasi-operasi tumpukan

```
int isEmpty(Stack *stack) {
```

Deklarasi fungsi **isEmpty** yang mengambil satu argument, yaitu pointer ke tumpukan (**Stack \*stack**) dan mengembalikan nilai integer

```
return stack->top == -1;
```

Implementasi fungsi **isEmpty** yang memeriksa apakah indeks **top** dari tumpukan sama dengan -1. Jika **top** sama dengan -1, berarti tumpukan kosong dan tidak ada elemen di dalamnya. Jadi, fungsi mengembalikan -1 jika tumpukan kosong, dan 0 jika tumpukan tidak kosong

```
int isFull(Stack *stack) {
```

Deklarasi fungsi **isFull** dengan pengembalian integer dan menerima satu parameter, yaitu pointer ke tumpukan (**Stack \*stack**)

```
return stack->top == stack->size - 1;
```

Pernyataan **return** digunakan untuk mengembalikan hasil pengecekan kondisi. Ekspresi **stack->top == stack->size - 1** akan menghasilkan nilai 1 jika nilai dari **top** dalam tumpukan sama dengan **size - 1**, yang menunjukkan bahwa tumpukan penuh. Sebaliknya, jika **top** tidak sama dengan **size - 1**, maka akan mengembalikan nilai 0 yang menunjukkan bahwa tumpukan belum penuh

```
void push(Stack *stack, char item) {
```

Deklarasi fungsi **push** yang menerima dua argument yaitu **stack** yang merupakan pointer ke tumpukan, dan **item** yang merupakan karakter yang akan ditambahkan ke dalam tumpukan. Fungsi ini tidak mengembalikan nilai (void)

```
if (!isFull(stack)) {
```

Memeriksa apakah tumpukan sudah penuh atau belum. Jika tumpukan belum penuh (**isFull(stack)** yang berarti belum penuh), maka dapat menambahkan elemen ke dalam tumpukan

```
stack->items[++stack->top] = item;
```

Jika tumpukan belum penuh, elemen baru dapat ditambahkan ke dalam tumpukan. **stack->top** menunjukkan posisi teratas tumpukan saat ini. **++stack->top** bertugas meningkatkan nilai **top** dengan 1 yang berarti akan berpindah ke posisi yang tepat diatas elemen teratas sebelumnya sehingga dapat menambahkan item baru. **stack->items[]** adalah array yang digunakan untuk menyimpan elemen-elemen tumpukan. Jadi, fungsi ini digunakan untuk berpindah “maju” ke posisi tepat diatas elemen teratas sebelumnya dengan cara menaikkan nilai **top** kemudian memasukkan **item** ke dalam posisi tersebut

```
} else {  
    printf("Stack penuh. Tidak bisa menambahkan elemen.\n");  
}
```

Jika tumpukan sudah penuh, program akan mencetak pesan ke layar dan memberi tahu pengguna bahwa tumpukan sudah penuh dan tidak bisa menampung elemen baru lagi

```
char pop(Stack *stack) {
```

Fungsi **pop** berfungsi untuk menghapus dan mengembalikan elemen teratas dari tumpukan dengan cara menerima tumpukan sebagai argument, yang diberikan dalam bentuk pointer (**Stack \*stack**), mengakses elemen teratas dari tumpukan, mengurangi nilai indeks elemen teratas sehingga elemen teratas berubah, dan mengembalikan nilai dari elemen teratas yang sudah dihapus

```
return stack->items[stack->top--];
```

Baris ini bertujuan mengembalikan nilai elemen teratas tumpukan dan menghapusnya dari tumpukan, sehingga tumpukan berkurang ukurannya

```
char isPair(char opening, char closing) {  
    return (opening == '(' && closing == ')') || (opening == '{' && closing ==  
    '}') || (opening == '[' && closing == ']');
```

```
}
```

Deklarasi fungsi **isPair** yang berguna memeriksa apakah sepasang tanda kurung berpasangan dan cocok. Baris ini memeriksa apakah karakter pembuka **opening** cocok dengan karakter penutup **closing**. Jika **opening** adalah tanda kurung buka, dan **closing** adalah tanda kurung tutup yang sesuai, maka akan menghasilkan 1 atau true. Jika tidak, maka akan mengembalikan 0 atau false.

```
char isBalanced(char *exp) {
```

Fungsi **isBalanced** untuk memeriksa apakah ekspresi matematika atau logika yang diberikan memiliki tanda kurung yang seimbang atau tidak

```
Stack *stack = createStack(strlen(exp));
```

Membuat tumpukan baru dengan fungsi **createStack** yang memiliki ukuran yang sama dengan panjang ekspresi yang diberikan

```
int i;
```

Variabel **i** dideklarasikan sebagai indeks yang akan digunakan dalam loop untuk menelusuri setiap karakter dalam ekspresi

```
for (i = 0; exp[i]; i++) {
```

Baris ini adalah awal dari loop **for**. Loop ini akan berjalan dari **i = 0** hingga **i** mencapai indeks terakhir dalam string **exp**. **exp[i]** digunakan sebagai kondisi loop, sehingga loop akan berhenti ketika mencapai karakter nul (\0) di akhir string

```
if (exp[i] == '(' || exp[i] == '{' || exp[i] == '[') {
```

Pada iterasi loop memeriksa karakter **exp[i]** untuk melihat apakah itu merupakan tanda buka kurung, kurawal, atau kurung siku

```
push(stack, exp[i]);
```

Jika karakter **exp[i]** adalah tanda buka kurung, kurawal, atau kurung siku, maka karakter tersebut akan dimasukkan ke dalam tumpukan menggunakan fungsi **push**

```
} else if (exp[i] == ')' || exp[i] == '}' || exp[i] == ']') {
```

Jika karakter **exp[i]** bukan merupakan tanda buka kurung, kurawal, atau kurung siku, maka akan memeriksa apakah itu merupakan tanda tutup kurung, kurawal, atau kurung siku

```
if (isEmpty(stack) || !isPair(pop(stack), exp[i])) {
```

Jika tumpukan kosong atau pasangan kurung tidak sesuai, maka ekspresi tersebut tidak seimbang. Jadi akan mengembalikan 0 dan membebaskan memori yang digunakan untuk tumpukan sebelumnya

```
free(stack->items);
    free(stack);
    return 0;
}
```

Membebaskan memori yang dialokasikan untuk array **items** dalam tumpukan serta tumpukan itu sendiri karena sudah selesai menggunakan mereka dalam perhitungan keseimbangan tanda kurung

```
int result = isEmpty(stack);
```

Memanggil fungsi **isEmpty** untuk memeriksa apakah tumpukan **stack** kosong atau tidak. Hasilnya berupa 1 jika tumpukan kosong dan 0 jika tidak, disimpan dalam variabel **result**

```
free(stack->items);
```

Setelah selesai menggunakan tumpukan **stack**, maka perlu membebaskan memori yang dialokasikan untuk array **items** dalam tumpukan. dilakukan dengan menggunakan fungsi **free** yang akan mengembalikan memori tersebut ke sistem operasi agar bisa digunakan lagi

```
free(stack);
```

Membebaskan memori yang dialokasikan untuk tumpukan itu sendiri menggunakan **free**

```
return result;
```

Mengembalikan nilai **result** dari fungsi. Ini adalah hasil dari pengecekan apakah tumpukan kosong atau tidak yang telah dilakukan sebelumnya. Mengembalikan 1 jika tumpukan kosong dan 0 jika tidak kosong. Ini akan memberi tahu pemanggil fungsi apakah ekspresi memiliki keseimbangan tanda kurung atau tidak

```
int main() {
```

Fungsi **main** adalah titik masuk utama dari program

```
char expression[100];
```

Mendeklarasikan sebuah array karakter bernama **expression** dengan ukuran 100. Array ini akan digunakan untuk menyimpan ekspresi yang dimasukkan oleh pengguna



```
printf("Masukkan set tanda kurung: ");
```

Mencetak pesan ke layar untuk meminta pengguna memasukkan ekspresi yang ingin diperiksa keseimbangannya

```
scanf("%s", expression);
```

Menggunakan **scanf** untuk membaca ekspresi yang dimasukkan oleh pengguna dan menyimpannya dalam array **expression**

```
for (int i = 0; expression[i]; i++) {
```

Menggunakan loop for untuk menelusuri setiap karakter dalam ekspresi yang dimasukkan oleh pengguna

```
if (!strchr("(){}[]", expression[i])) {
```

Memeriksa setiap karakter dalam ekspresi untuk memastikan bahwa hanya karakter tanda kurung yang valid ( (){}[] ) yang dimasukkan oleh pengguna

```
printf("Ekspresi tidak valid.\n");
```

```
    return 0;
```

Jika ditemukan karakter selain tanda kurung, program akan mencetak pesan kesalahan dan program selesai dieksekusi

```
printf("%s\n", isBalanced(expression) ? "YES" : "NO");
```

Setelah memastikan bahwa ekspresi yang dimasukkan oleh pengguna valid, kemudian memanggil fungsi **isBalanced** untuk memeriksa apakah ekspresi tersebut memiliki keseimbangan tanda kurung atau tidak. Hasil dari pengecekan tersebut akan dicetak ke layar, yaitu “YES” jika ekspresi seimbang dan “NO” jika tidak

```
return 0;
```

Mengembalikan nilai 0 untuk menandakan bahwa program berjalan dengan sukses dan berhasil menyelesaikan eksekusi