PRAKTIKUM STRUKTUR DATA

INF11198

2401020010 – FACHREZI BACHRI

1. Berikut source code yang sudah saya modifikasi dimana code ini mensimulasikan operasi **Stack**. Saya hanya lengkapi kekurangan source code yang di berikan tanpa merubah yang sudah ada.

   Sourch code yang sudah di modifikasi:

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

#define MAX 100 // Maximum size of the stack

// Stack structure
typedef struct Stack {
  int data[MAX];
  int top;
} Stack;

// Initialize the stack
void initialize(Stack *stack) {
  stack->top = -1;
}

// Check if the stack is empty
int isEmpty(Stack *stack) {
  return stack->top == -1;
}
```

```
// Check if the stack is full
int isFull(Stack *stack) {
  return stack->top == MAX - 1;
}

// Push an element onto the stack
void push(Stack *stack, int value) {
  if (isFull(stack)) {
    printf("Stack Overflow! Cannot push %d\n", value);
    return;
  }
  stack->data[++stack->top] = value;
  printf("Pushed %d onto the stack.\n", value);
}

// Pop an element from the stack
int pop(Stack *stack) {
  if (isEmpty(stack)) {
    printf("Stack Underflow! Cannot pop.\n");
    return -1; // Return -1 to indicate an error
  }
  return stack->data[stack->top--];
}

// Peek at the top element of the stack
int peek(Stack *stack) {
  if (isEmpty(stack)) {
    printf("Stack is empty! Cannot peek.\n");
    return -1; // Return -1 to indicate an error
  }
  return stack->data[stack->top];
```

```
}

// Main function to demonstrate stack operations
int main() {
  Stack stack;
  initialize(&stack);

  push(&stack, 10);
  push(&stack, 20);
  push(&stack, 30);

  printf("Top element is: %d\n", peek(&stack));

  printf("Popped element: %d\n", pop(&stack));
  printf("Popped element: %d\n", pop(&stack));

  printf("Is stack empty? %s\n", isEmpty(&stack) ? "Yes" : "No");

  return 0;
}
```

2. Berikut source code yang sudah saya modifikasi dimana code ini berguna untuk operasi stack supaya berhungsi. Saya menambahkan kekurangan code tanpa merubah isi baris fungsi main.

   Sourch code yang sudah saya lengkapi:

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

char **stackString;
int  pStack;

int push(char *kata){
  if (stackString == NULL){
    stackString = (char **) malloc(sizeof(char **));
    stackString[pStack] = (char *) malloc(sizeof(char *));
    printf("success created stack\n");
  }
   char  **  stackStringT  =  (char  **)  realloc(stackString,
(pStack+1)*sizeof(char *));
  if (stackStringT){
    stackString = stackStringT;
    stackString[pStack] = (char *) malloc(strlen(kata) + 1);
  }
  if (strcpy(stackString[pStack], kata) != NULL){
    pStack++;
    return 1;
  }
  else
    return 0;
}
```

```
int pop(){
  if (stackString == NULL){
    printf("No stack found\n");
    return 0;
  }
  if (pStack!=0) {
        printf("\033[1;92mPop    =>\033[37;44m    %s    \033[0m\n",
stackString[--pStack]);
    free(stackString[pStack]);
    return 1;
  } else {
    printf("\033[1;31mStack Empty \n");
    return 0;
  }
}

void popAll(){
  while (pop()){

  }
  free(stackString);
}

void indirect(void (*func) ()) {
    func();
}

// main fuction. Do not Modify !!!!
int main(int argc, char **params){
  pStack = 0;
  if (argc <= 1){
```

```
      printf("\e[5mError.\n\e[1mFormat  :\n\t\e[3m./stack.o  kata1
kata2 kata3 ...\n\e[0m");
    return 1;
  }

  for (int i=1; i<argc;i++){
    if (!push(params[i])){
      printf("Push Operation %d Out of memory\n", i);
    }

  }

  // Here is the advance technique using pointer to call function
without explicitly
  // using the function name, rather than only use the pointer
variable
  void (*pr) () = popAll;
  indirect( pr );

  return 0;
}
```

3. Berikut source code yang sudah saya modifikasi dimana code ini mensimulasikan operasi **Queue**. Saya hanya lengkapi kekurangan source code yang di berikan tanpa merubah yang sudah ada.

   Sourch code yang sudah di modifikasi:

```
#include <stdio.h>
#define SIZE 5 // Maximum size of the queue

int queue[SIZE];
int front = -1, rear = -1;

// Fuction to add an element to the queue
void enqueue(int value) {
  if (rear == SIZE - 1) {
    printf("Queue is full! Cannot enqueue %d.\n", value);
  } else {
// Initialize front if it's the first element
    if (front == -1) front = 0;
    queue[++rear] = value;
    printf("Enqueued: %d\n", value);
  }
}

// Function to remove an element from the queue
void dequeue() {
  if (front == -1) {
    printf("Queue is empty! Cannot dequeue.\n");
  } else {
    printf("Dequeued: %d\n", queue[front++]);
    if (front > rear) front = rear = -1; // Reset queue if empty
  }
```

```
}

// Function to display the queue
void display() {
  if (front == -1) {
    printf("Queue is empty!\n");
  } else {
    printf("Queue elements: ");
    for (int i = front; i <= rear; i++) {
       printf("%d ", queue[i]);
    }
    printf("\n");
  }
}

// Main fuction to demonstrate queue operations
int main(){
  enqueue(10);
  enqueue(20);
  enqueue(30);
  display();
  dequeue();
  display();
  enqueue(40);
  enqueue(50);
  enqueue(60);  // This will show "Queue is full"
  display();
  return 0;
}
```

4. Berikut source code yang sudah saya modifikasi dimana code ini mensimulasikan operasi **Priority Queue**. Saya hanya lengkapi kekurangan source code yang di berikan tanpa merubah yang sudah ada.

   Sourch code yang sudah di modifikasi:

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

#define MAX 100

int priorityQueue[MAX];
int size = 0;

// Fuction to insert an element into the prority queue
void insert(int value) {
  if (size >= MAX) {
    printf("Priority Queue is full!\n");
    return;
  }
  size++;
  int i = size;
  while (i > 1 && value > priorityQueue[i / 2]) {
    priorityQueue[i] = priorityQueue[i / 2];
    i /= 2;
  }
  priorityQueue[i] = value;
}

// Fuction to delete the highest priority element
void deleteMax() {
  if (size == 0) {
```

```
    printf("Priority Queue is empty!\n");
    return;
  }
  int max = priorityQueue[1];
  int last = priorityQueue[size];
  size--;
  int i = 1, child;
  while (i * 2 <= size) {
    child = i * 2;
      if (child + 1 <= size && priorityQueue[child + 1] > priorityQueue[child]) {
       child++;
    }
    if (last >= priorityQueue[child]) break;
    priorityQueue[i] = priorityQueue[child];
    i = child;
  }
  priorityQueue[i] = last;

  printf("Deleted element: %d\n", max);
}

// Function to display the priority queue
void display() {
  if (size == 0) {
    printf("Priority Queue is empty!\n");
    return;
  }
  printf("Priority Queue: ");
  for (int i = 1; i <= size; i++) {
     printf("%d ", priorityQueue[i]);
  }
```

```
  printf("\n");
}

// Main fuction to demonstrate the operations
int main() {
   int choice, value;

   while (1) {
     printf("\nPriority Queue Operations:\n");
     printf("1. Insert\n");
     printf("2. Delete Max\n");
     printf("3. Display\n");
     printf("4. Exit\n");
     printf("Enter your choice: ");
     scanf("%d", &choice);

     switch (choice) {
       case 1:
         printf("Enter value to insert: ");
         scanf("%d", &value);
         insert(value);
         break;
       case 2:
         deleteMax();
         break;
       case 3:
         display();
         break;
       case 4:
         exit(0);
       default:
         printf("Invalid choice! Please try again.\n");
```

```
        }
    }
    return 0;
}
```