CS 2263 - FR01A

Assignment 2

By Ngoc Phuong Anh Nguyen – 3712361

October 2021

Questions:

```
/** … */
bool push(int *stack, int *size, int max_size, int to_push)
{
  if(*size < max_size)
  {
    *(stack+*size) = to_push;
    *size = *size +1;
    return true;
  }
  else
  {
    return false;
  }
}

/** … */
bool pop(int *stack, int *size, int *to_return)
{
  if(*size > 0)
  {
    *to_return = *(stack+*size-1);
    *size = *size -1;
    return true;
  }
  else
  {
    return false;
  }
}
```

Figure 1: push and pop function

```
bool peek(int *stack, int *size, int *to_return)
{
  if(*size)
  {
    *to_return = *(stack+*size-1);
    return true;
  }
  else
  {
    return false;
  }
}
```

Figure 2: Peek function

```c
int main( int argc, char **argv )
{
  // keep track of the max size and the current size of the stack
  int stack_max_size = STACK_MAX_SIZE;
  int stack_current_size = 0;
  int i;

  // the stack is an array located on the main() function stack frame
  int stack[stack_max_size];

  // initialize our stack with 0 values
  for(i=0; i < stack_max_size; i++)
  {
    stack[i] = 0;
  }

  // count the number of instructions (peek, pop, push) that successfully happened
  int successful_instructions = 0;
  bool stop_execution = false;

  while(!stop_execution)
  {
    // read the input instruction (a single character)
    char input_instruction = 0;
    fflush(stdin);
    scanf("%c", &input_instruction);
    int value;
    // the character could be a whitespace so we need to skip those
    if(false == is_whitespace(input_instruction))
    {
      if(input_instruction == 'x')
```

Figure 3: main function

```
if(input_instruction == 'x')
{
  stop_execution = true;
}
else if(input_instruction == 'u')
{
  scanf("%d", &value);
  bool pushStack = push(stack, &stack_current_size, stack_max_size, value);
  if(pushStack == true)
  {
    printf("%d\n", value);
    successful_instructions++;
  }
  else
  {
    printf("failed push\n");
  }
  int i;
}
else if(input_instruction == 'o')
{
  bool popStack = pop(stack, &stack_current_size, &value);
  if(popStack == true)
  {
    printf("%d\n", value);
    successful_instructions++;
  }
  else
  {
    printf("failed pop\n");
  }
}
else if(input_instruction == 'e')
{
  bool peekStack = peek(stack, &stack_current_size, &value);
  if(peekStack == true)
```

Figure 4: main function

```
        if(peekStack == true)
        {
          printf("%d\n", value);
          successful_instructions++;
        }
        else
        {
          printf("failed peek\n");
        }
      }
      else
      {
        printf("invalid instruction %c\n", input_instruction);
      }
    }
  }

  printf("Successfully executed %d instructions\n", successful_instructions);
  print_stack(stack, &stack_current_size);

  return EXIT_SUCCESS;
}
```

Figure 5: main function

```
[anguyen5@gc112m30 ~/A2src]$ make Stack
gcc -Wall -Wextra -c Stack.c
Stack.c: In function ‘main’:
Stack.c:241:9: warning: unused variable ‘i’ [-Wunused-variable]
     int i;
         ^
Stack.c:194:15: warning: unused parameter ‘argc’ [-Wunused-parameter]
 int main( int argc, char **argv )
               ^
Stack.c:194:28: warning: unused parameter ‘argv’ [-Wunused-parameter]
 int main( int argc, char **argv )
                            ^
gcc -Wall -Wextra -o Stack Stack.o
[anguyen5@gc112m30 ~/A2src]$ make test
./Stack < Data/exit_test1.input > exit_test1.result
./TestPassed.sh exit_test1.result Data/exit_test1.expected

######   Passed   ###### exit_test1.result is equal to Data/exit_test1.expected

./Stack < Data/push_test1.input > push_test1.result
./TestPassed.sh push_test1.result Data/push_test1.expected

######   Passed   ###### push_test1.result is equal to Data/push_test1.expected

./Stack < Data/push_test2.input > push_test2.result
./TestPassed.sh push_test2.result Data/push_test2.expected

######   Passed   ###### push_test2.result is equal to Data/push_test2.expected

./Stack < Data/peek_test1.input > peek_test1.result
./TestPassed.sh peek_test1.result Data/peek_test1.expected

######   Passed   ###### peek_test1.result is equal to Data/peek_test1.expected

./Stack < Data/peek_test2.input > peek_test2.result
./TestPassed.sh peek_test2.result Data/peek_test2.expected

######   Passed   ###### peek_test2.result is equal to Data/peek_test2.expected

./Stack < Data/pop_test1.input > pop_test1.result
./TestPassed.sh pop_test1.result Data/pop_test1.expected

######   Passed   ###### pop_test1.result is equal to Data/pop_test1.expected

./Stack < Data/pop_test2.input > pop_test2.result
./TestPassed.sh pop_test2.result Data/pop_test2.expected

######   Passed   ###### pop_test2.result is equal to Data/pop_test2.expected

./Stack < Data/pop_test3.input > pop_test3.result
./TestPassed.sh pop_test3.result Data/pop_test3.expected
```

```
./TestPassed.sh push_test2.result Data/push_test2.expected

######   Passed   ###### push_test2.result is equal to Data/push_test2.expected

./Stack < Data/peek_test1.input > peek_test1.result
./TestPassed.sh peek_test1.result Data/peek_test1.expected

######   Passed   ###### peek_test1.result is equal to Data/peek_test1.expected

./Stack < Data/peek_test2.input > peek_test2.result
./TestPassed.sh peek_test2.result Data/peek_test2.expected

######   Passed   ###### peek_test2.result is equal to Data/peek_test2.expected

./Stack < Data/pop_test1.input > pop_test1.result
./TestPassed.sh pop_test1.result Data/pop_test1.expected

######   Passed   ###### pop_test1.result is equal to Data/pop_test1.expected

./Stack < Data/pop_test2.input > pop_test2.result
./TestPassed.sh pop_test2.result Data/pop_test2.expected

######   Passed   ###### pop_test2.result is equal to Data/pop_test2.expected

./Stack < Data/pop_test3.input > pop_test3.result
./TestPassed.sh pop_test3.result Data/pop_test3.expected

######   Passed   ###### pop_test3.result is equal to Data/pop_test3.expected

./Stack < Data/compound_test1.input > compound_test1.result
./TestPassed.sh compound_test1.result Data/compound_test1.expected

######   Passed   ###### compound_test1.result is equal to Data/compound_test1.expected

./Stack < Data/compound_test2.input > compound_test2.result
./TestPassed.sh compound_test2.result Data/compound_test2.expected

######   Passed   ###### compound_test2.result is equal to Data/compound_test2.expected

./Stack < Data/compound_test3.input > compound_test3.result
./TestPassed.sh compound_test3.result Data/compound_test3.expected

######   Passed   ###### compound_test3.result is equal to Data/compound_test3.expected

./Stack < Data/newtest_compound.input > newtest_compound.result
./TestPassed.sh newtest_compound.result Data/newtest_compound.expected

######   Passed   ###### newtest_compound.result is equal to Data/newtest_compound.expected

[anguyen5@gc112m30 ~/A2src]$
```