INDIVIDUAL ASSIGNMENT

TECHNOLOGY PARK MALAYSIA

CT018-3-1-ICP

INTRODUCTION TO C PROGRAMMING

APU-APD1F2009/IT/CS(DA)/CGD/CS/CS(CYB)/CS(IS)/ TE/PE/CE/EEE/ME

STUDENT NAME: RYAN MARTIN

TP NUMBER: TP058091

HAND OUT DATE: 3 JULY 2021

HAND IN DATE: 28 JUNE 2021

WEIGHTAGE: 50%

Table of Contents

Introduction........ 1
Assumptions........ 1
Program Design (Pseudocode & Flowchart) 2
Sample Input and Outputs........ 42
Conclusion 54

Introduction

Right now, the world is facing a pandemic caused by the SARS-COV-2 virus, or more commonly known as the coronavirus. This virus targets the human respiratory system and can cause a disease called COVID-19. Many vaccines are being developed in order to counter this virus.

In Malaysia, the vaccination process for citizens has already started. Many hospitals and health care have started giving out vaccines to people. For this assignment, we are tasked to create a vaccine inventory management system for these healthcare providers. The system needs to have the following functionalities:

1. Inventory creation: The system should allow employees to record vaccine details into the file ‘vaccines.txt’. The vaccines have initial quantities defined by the programmer.
2. Update vaccine quantities: The system should allow employees to update the quantity of each vaccine. If vaccine is being distributed, log the vaccine code and amount in the file ‘dist.txt’.
3. Search vaccine and its available quantity by its code: Employees should be able to check the available quantity of each vaccine by entering its code.
4. Produce a list of vaccines and their distributed quantities: The system should provide a summary of the distributed quantities of each vaccine in ‘dist.txt’, sorted in descending order.

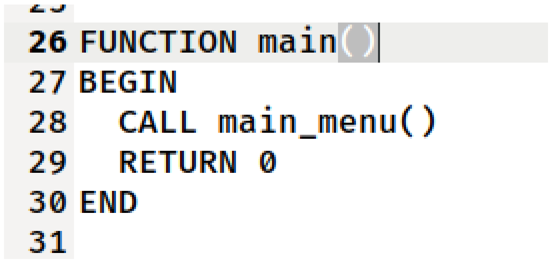
Assumptions

Below are the assumptions I made about the program:

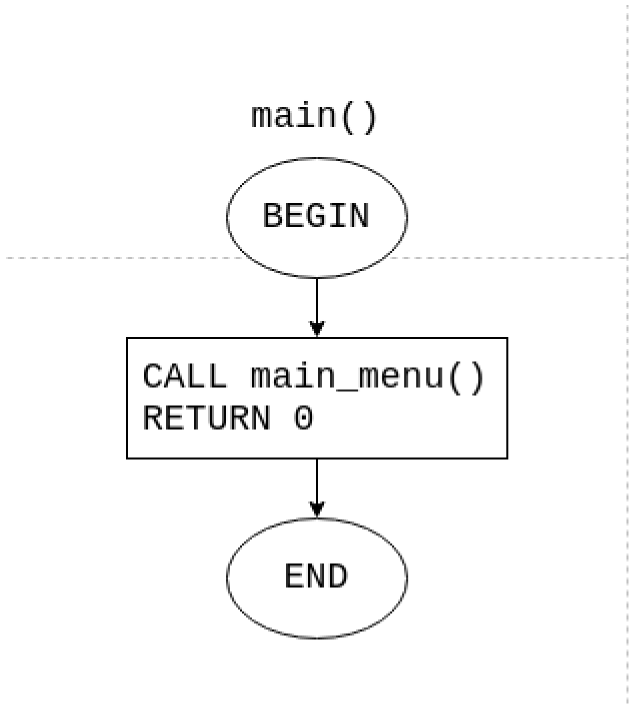
1. Since the initial quantities of the vaccines are decided by the programmer, employees should not have to manually enter each of the vaccines and their details themselves. This is done automatically by the program.
2. Since the question description did not mention anything related to adding more types of vaccine provided, there is no functionality for it in the system.
3. Only selected employees would be able to access this system, so no protection in the form of user authentication is provided in the system.
4. The unit for vaccine quantity is millions. Adding and removing vaccines will be in the millions, so that the value stays integer.

Program Design (Pseudocode & Flowchart)

1. main function

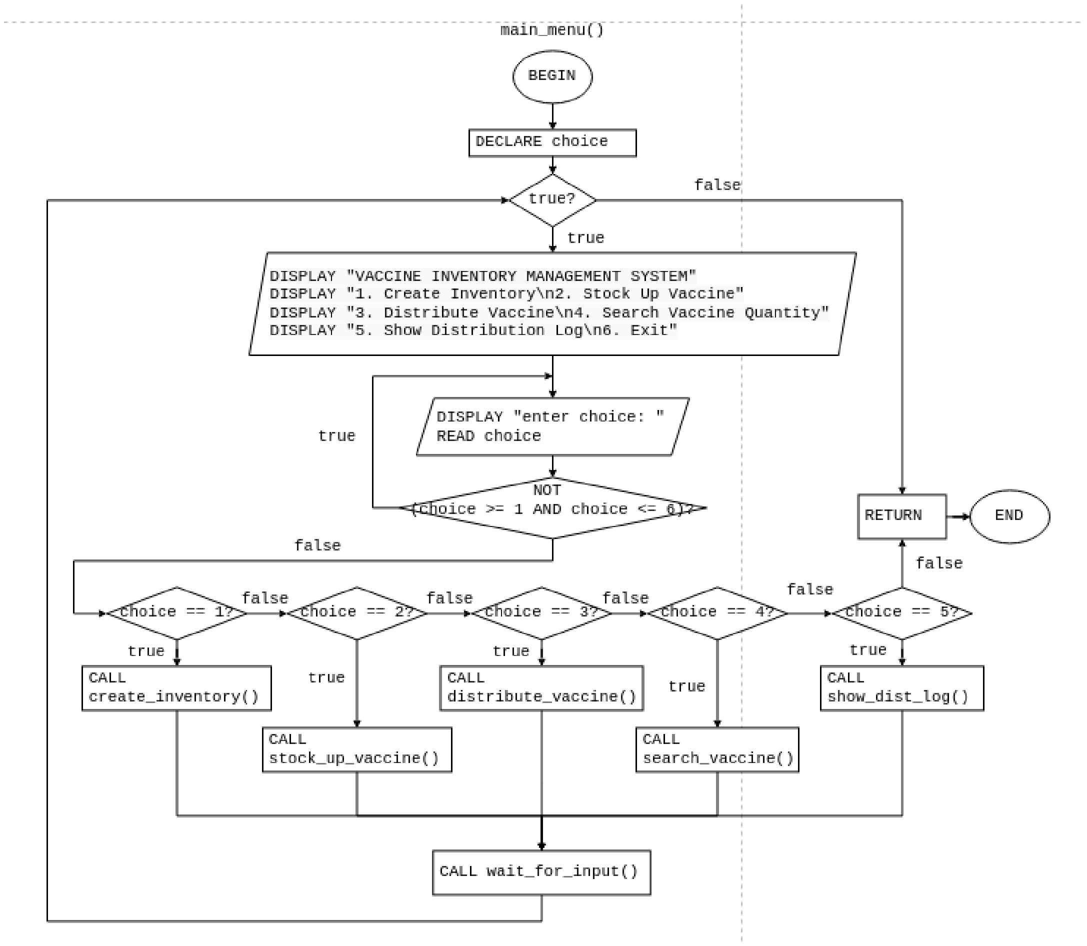


```
FUNCTION main()
BEGIN
  CALL main_menu()
  RETURN 0
END
```



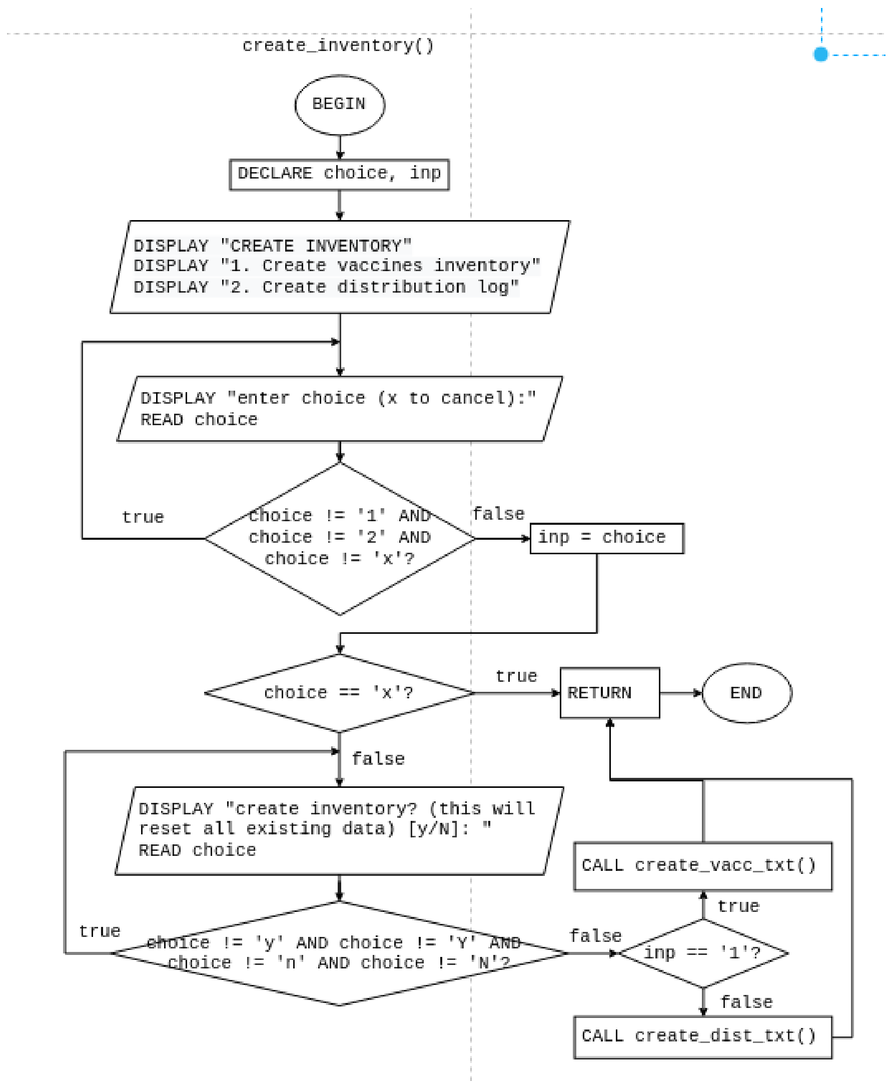
2. main_menu function

```
FUNCTION main_menu()
BEGIN
  DECLARE choice
  WHILE true
    DISPLAY "VACCINE INVENTORY MANAGEMENT SYSTEM"
    DISPLAY "1. Create Inventory\n2. Stock Up Vaccine"
    DISPLAY "3. Distribute Vaccine\n4. Search Vaccine Quantity"
    DISPLAY "5. Show Distribution Log\n6. Exit"
    DO
      DISPLAY "enter choice"
      READ choice
    WHILE NOT (choice ⩾ 1 AND choice ⩽ 6)
    IF choice == 1 THEN
      CALL create_inventory()
    ELSE IF choice == 2 THEN
      CALL stock_up_vaccine()
    ELSE IF choice == 3 THEN
      CALL distribute_vaccine()
    ELSE IF choice == 4 THEN
      CALL search_vaccine()
    ELSE IF choice == 5 THEN
      CALL show_dist_log()
    ELSE
      RETURN
    ENDIF
    CALL wait_for_input()
  ENDWHILE
END
```



3. create_inventory function

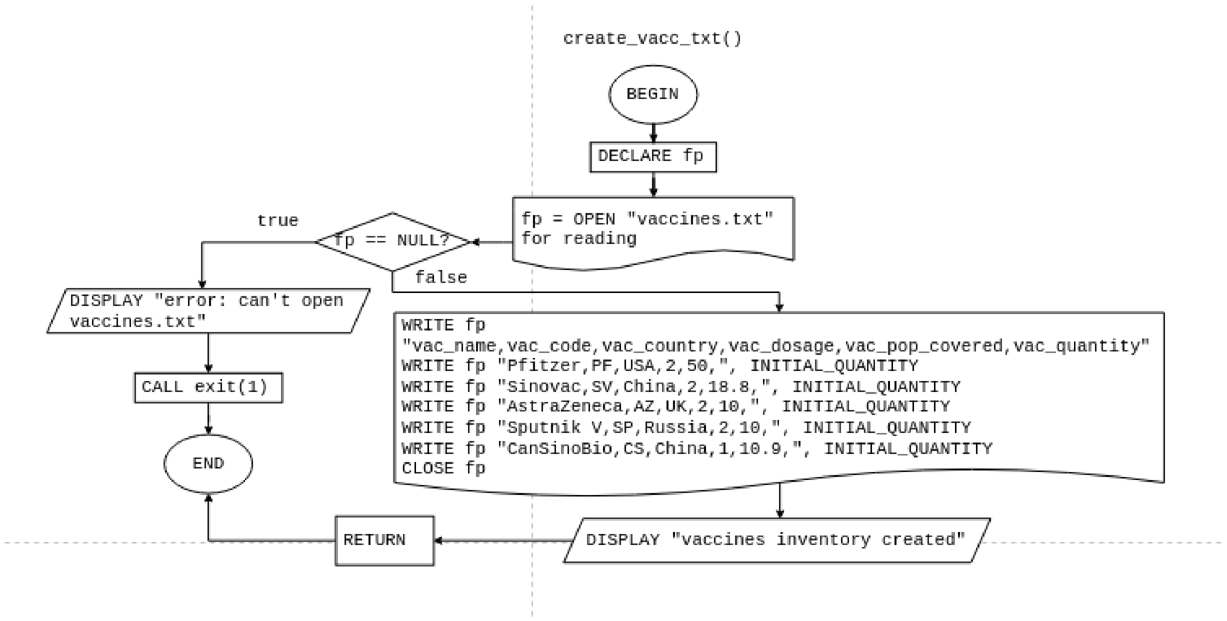
```
FUNCTION create_inventory()
BEGIN
  DECLARE choice, inp
  DISPLAY "CREATE INVENTORY"
  DISPLAY "1. Create vaccines inventory\n2. Create distribution log"
  DO
    DISPLAY "enter choice (x to cancel): "
    READ choice
  WHILE choice ≠ '1' AND choice ≠ '2' AND choice ≠ 'x'
  inp = choice
  IF choice == 'x' THEN
    RETURN
  ENDIF
  DO
    DISPLAY "create inventory? (this will reset all existing data) [y/N]: "
    READ choice
  WHILE choice ≠ 'y' AND choice ≠ 'Y' AND choice ≠ 'n' AND choice ≠ 'N'
  IF choice == 'y' OR choice == 'Y' THEN
    IF inp == '1' THEN
      CALL create_vacc_txt()
    ELSE
      CALL create_dist_txt()
    ENDIF
  ENDIF
END
```



create_inventory()
BEGIN
DECLARE choice, inp
DISPLAY "CREATE INVENTORY"
DISPLAY "1. Create vaccines inventory"
DISPLAY "2. Create distribution log"
DISPLAY "enter choice (x to cancel):"
READ choice
true
choice != '1' AND choice != '2' AND choice != 'x'?
false
inp = choice
choice == 'x'?
true
RETURN
END
false
DISPLAY "create inventory? (this will reset all existing data) [y/N]: "
READ choice
CALL create_vacc_txt()
true
true
choice != 'y' AND choice != 'Y' AND choice != 'n' AND choice != 'N'?
false
inp == '1'?
false
CALL create_dist_txt()

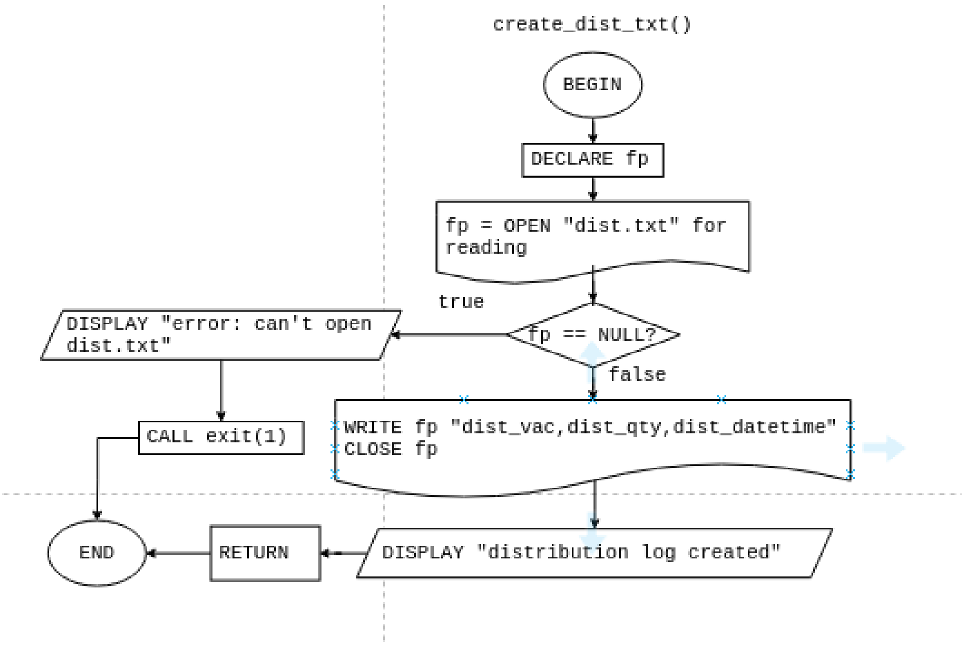
4. create_vacc_txt function

```
FUNCTION create_vacc_txt()
BEGIN
  DECLARE fp
  fp = OPEN "vaccines.txt" for reading
  IF fp == NULL THEN
    DISPLAY "error: can't open vaccines.txt"
    CALL exit(1)
  ENDIF
  WRITE fp "vac_name,vac_code,vac_country,vac_dosage,vac_pop_covered,vac_quantity"
  WRITE fp "Pfitzer,PF,USA,2,50,", INITIAL_QUANTITY
  WRITE fp "Sinovac,SV,China,2,18.8,", INITIAL_QUANTITY
  WRITE fp "AstraZeneca,AZ,UK,2,10,", INITIAL_QUANTITY
  WRITE fp "Sputnik V,SP,Russia,2,10,", INITIAL_QUANTITY
  WRITE fp "CanSinoBio,CS,China,1,10.9,", INITIAL_QUANTITY
  CLOSE fp
  DISPLAY "vaccines inventory created"
END
```


create_vacc_txt()
BEGIN
DECLARE fp
fp = OPEN "vaccines.txt" for reading
fp == NULL?
true
false
DISPLAY "error: can't open vaccines.txt"
CALL exit(1)
END
WRITE fp "vac_name,vac_code,vac_country,vac_dosage,vac_pop_covered,vac_quantity"
WRITE fp "Pfitzer,PF,USA,2,50,", INITIAL_QUANTITY
WRITE fp "Sinovac,SV,China,2,18.8,", INITIAL_QUANTITY
WRITE fp "AstraZeneca,AZ,UK,2,10,", INITIAL_QUANTITY
WRITE fp "Sputnik V,SP,Russia,2,10,", INITIAL_QUANTITY
WRITE fp "CanSinoBio,CS,China,1,10.9,", INITIAL_QUANTITY
CLOSE fp
DISPLAY "vaccines inventory created"
RETURN

5. create_dist_txt function

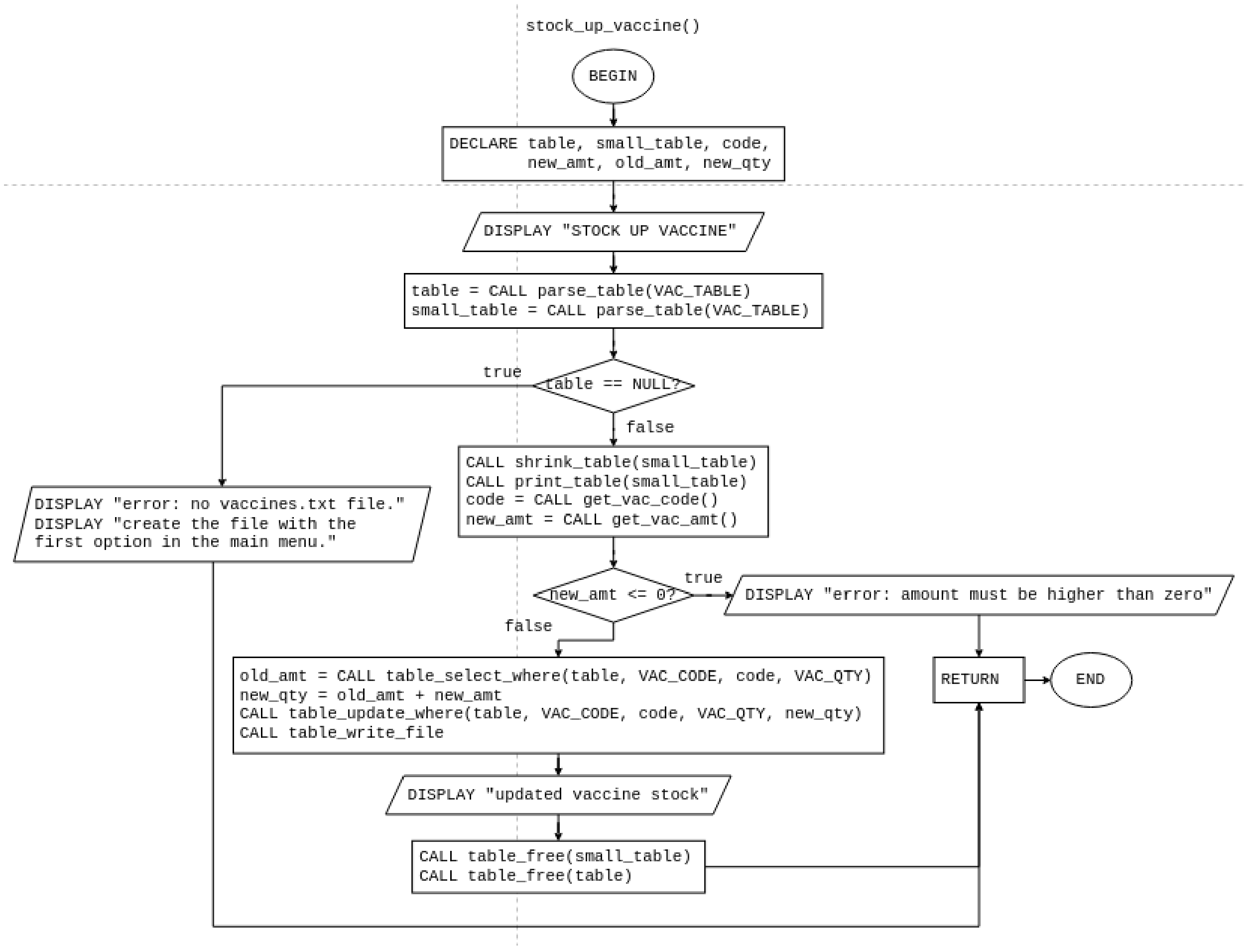
```

FUNCTION create_dist_txt()
BEGIN
  DECLARE fp
  fp = OPEN "dist.txt" for reading
  IF fp == NULL THEN
    DISPLAY "error: can't open dist.txt"
    CALL exit(1)
  ENDIF
  WRITE fp "dist_vac,dist_qty,dist_datetime"
  CLOSE fp
  DISPLAY "distribution log created"
END

```



create_dist_txt()
BEGIN
DECLARE fp
fp = OPEN "dist.txt" for reading
fp == NULL?
true
false
DISPLAY "error: can't open dist.txt"
CALL exit(1)
WRITE fp "dist_vac,dist_qty,dist_datetime"
CLOSE fp
DISPLAY "distribution log created"
RETURN
END

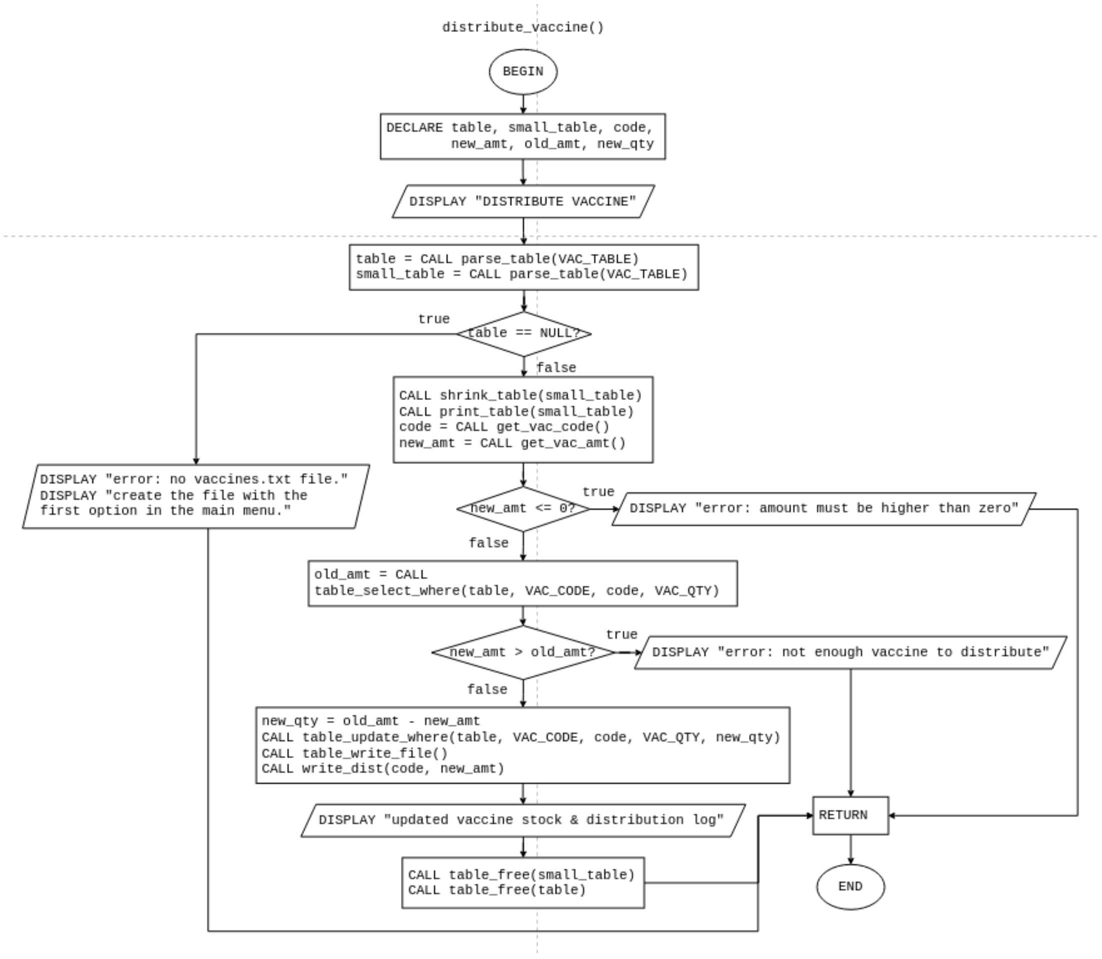
6. stock_up_vaccine function

```
FUNCTION stock_up_vaccine()
BEGIN
  DECLARE table, small_table, code, new_amt, old_amt, new_qty
  DISPLAY "STOCK UP VACCINE"
  table = CALL parse_table(VAC_TABLE)
  small_table = CALL parse_table(VAC_TABLE)
  IF table == NULL THEN
    DISPLAY "error: no vaccines.txt file."
    DISPLAY "create the file with the first option in the main menu."
    RETURN
  ENDIF
  CALL shrink_table(small_table)
  CALL print_table(small_table)
  code = CALL get_vac_code()
  new_amt = CALL get_vac_amt()
  IF new_amt ⩽ 0 THEN
    DISPLAY "error: amount must be higher than zero"
    RETURN
  ENDIF
  old_amt = CALL table_select_where(table, VAC_CODE, code, VAC_QTY)
  new_qty = old_amt + new_amt
  CALL table_update_where(table, VAC_CODE, code, VAC_QTY, new_qty)
  CALL table_write_file
  DISPLAY "updated vaccine stock"
  CALL table_free(small_table)
  CALL table_free(table)
END
```

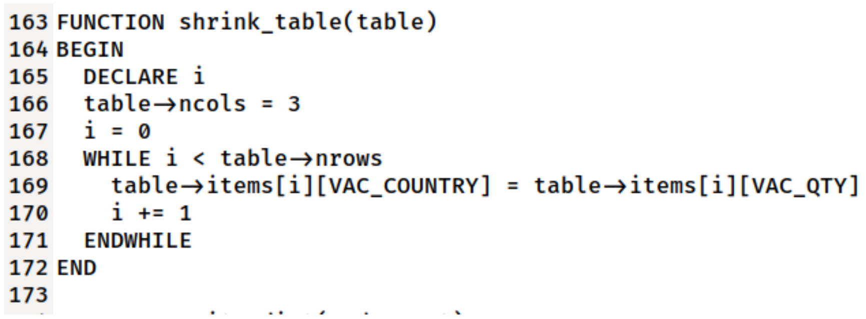

stock_up_vaccine()
BEGIN
DECLARE table, small_table, code, new_amt, old_amt, new_qty
DISPLAY "STOCK UP VACCINE"
table = CALL parse_table(VAC_TABLE)
small_table = CALL parse_table(VAC_TABLE)
table == NULL?
true
false
DISPLAY "error: no vaccines.txt file."
DISPLAY "create the file with the first option in the main menu."
CALL shrink_table(small_table)
CALL print_table(small_table)
code = CALL get_vac_code()
new_amt = CALL get_vac_amt()
new_amt <= 0?
true
false
DISPLAY "error: amount must be higher than zero"
old_amt = CALL table_select_where(table, VAC_CODE, code, VAC_QTY)
new_qty = old_amt + new_amt
CALL table_update_where(table, VAC_CODE, code, VAC_QTY, new_qty)
CALL table_write_file
DISPLAY "updated vaccine stock"
CALL table_free(small_table)
CALL table_free(table)
RETURN
END

7. distribute_vaccine function

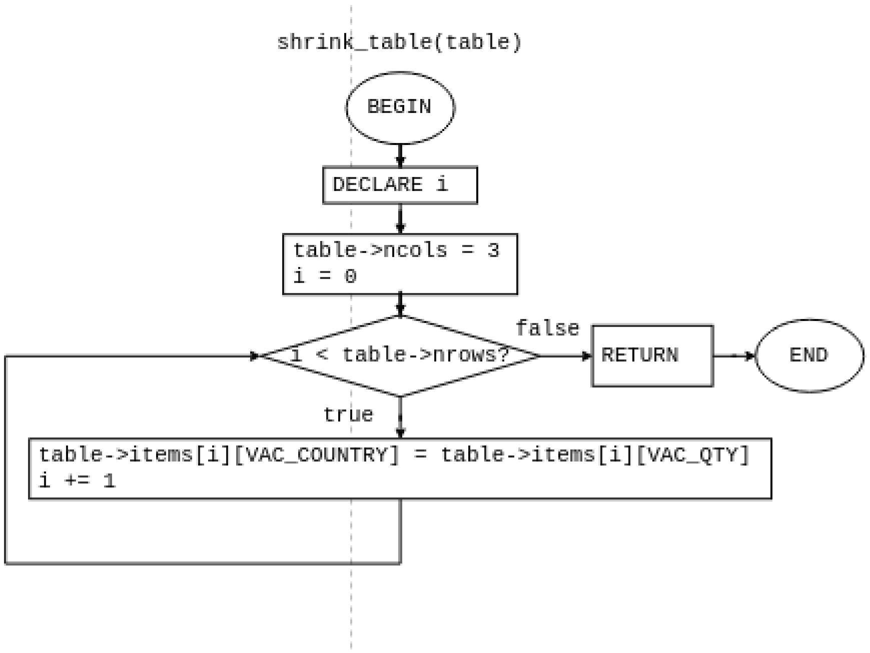
```
FUNCTION distribute_vaccine()
BEGIN
  DECLARE table, small_table, code, new_amt, old_amt, new_qty
  DISPLAY "DISTRIBUTE VACCINE"
  table = CALL parse_table(VAC_TABLE)
  small_table = CALL parse_table(VAC_TABLE)
  IF table == NULL THEN
    DISPLAY "error: no vaccines.txt file."
    DISPLAY "create the file with the first option in the main menu."
    RETURN
  ENDIF
  CALL shrink_table(small_table)
  CALL print_table(small_table)
  code = CALL get_vac_code()
  new_amt = CALL get_vac_amt()
  IF new_amt ≤ 0 THEN
    DISPLAY "error: amount must be higher than zero"
    RETURN
  ENDIF
  old_amt = CALL table_select_where(table, VAC_CODE, code, VAC_QTY)
  IF new_amt > old_amt THEN
    DISPLAY "error: not enough vaccine to distribute"
    RETURN
  ENDIF
  new_qty = old_amt - new_amt
  CALL table_update_where(table, VAC_CODE, code, VAC_QTY, new_qty)
  CALL table_write_file()
  CALL write_dist(code, new_amt)
  DISPLAY "updated vaccine stock & distribution log"
  CALL table_free(small_table)
  CALL table_free(table)
END
```


distribute_vaccine()
BEGIN
DECLARE table, small_table, code, new_amt, old_amt, new_qty
DISPLAY "DISTRIBUTE VACCINE"
table = CALL parse_table(VAC_TABLE)
small_table = CALL parse_table(VAC_TABLE)
table == NULL?
true
false
DISPLAY "error: no vaccines.txt file."
DISPLAY "create the file with the first option in the main menu."
CALL shrink_table(small_table)
CALL print_table(small_table)
code = CALL get_vac_code()
new_amt = CALL get_vac_amt()
new_amt <= 0?
true
DISPLAY "error: amount must be higher than zero"
false
old_amt = CALL table_select_where(table, VAC_CODE, code, VAC_QTY)
new_amt > old_amt?
true
DISPLAY "error: not enough vaccine to distribute"
false
new_qty = old_amt - new_amt
CALL table_update_where(table, VAC_CODE, code, VAC_QTY, new_qty)
CALL table_write_file()
CALL write_dist(code, new_amt)
DISPLAY "updated vaccine stock & distribution log"
CALL table_free(small_table)
CALL table_free(table)
RETURN
END

8. shrink_table function

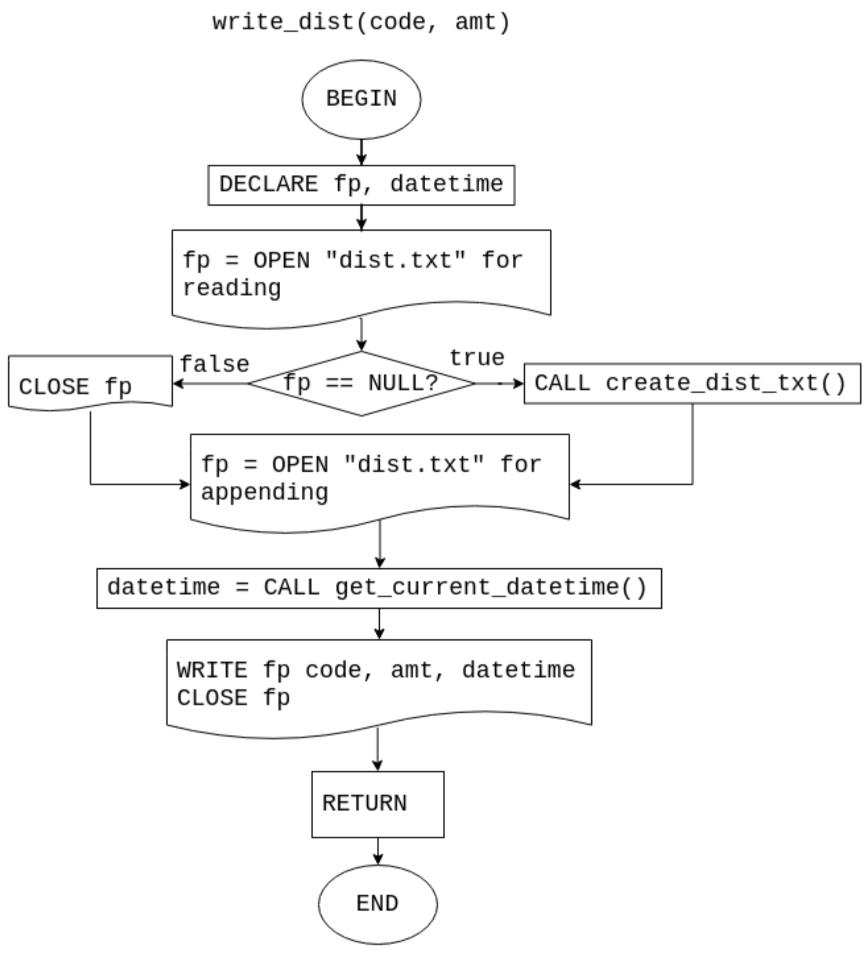


```
FUNCTION shrink_table(table)
BEGIN
  DECLARE i
  table→ncols = 3
  i = 0
  WHILE i < table→nrows
    table→items[i][VAC_COUNTRY] = table→items[i][VAC_QTY]
    i += 1
  ENDWHILE
END

```



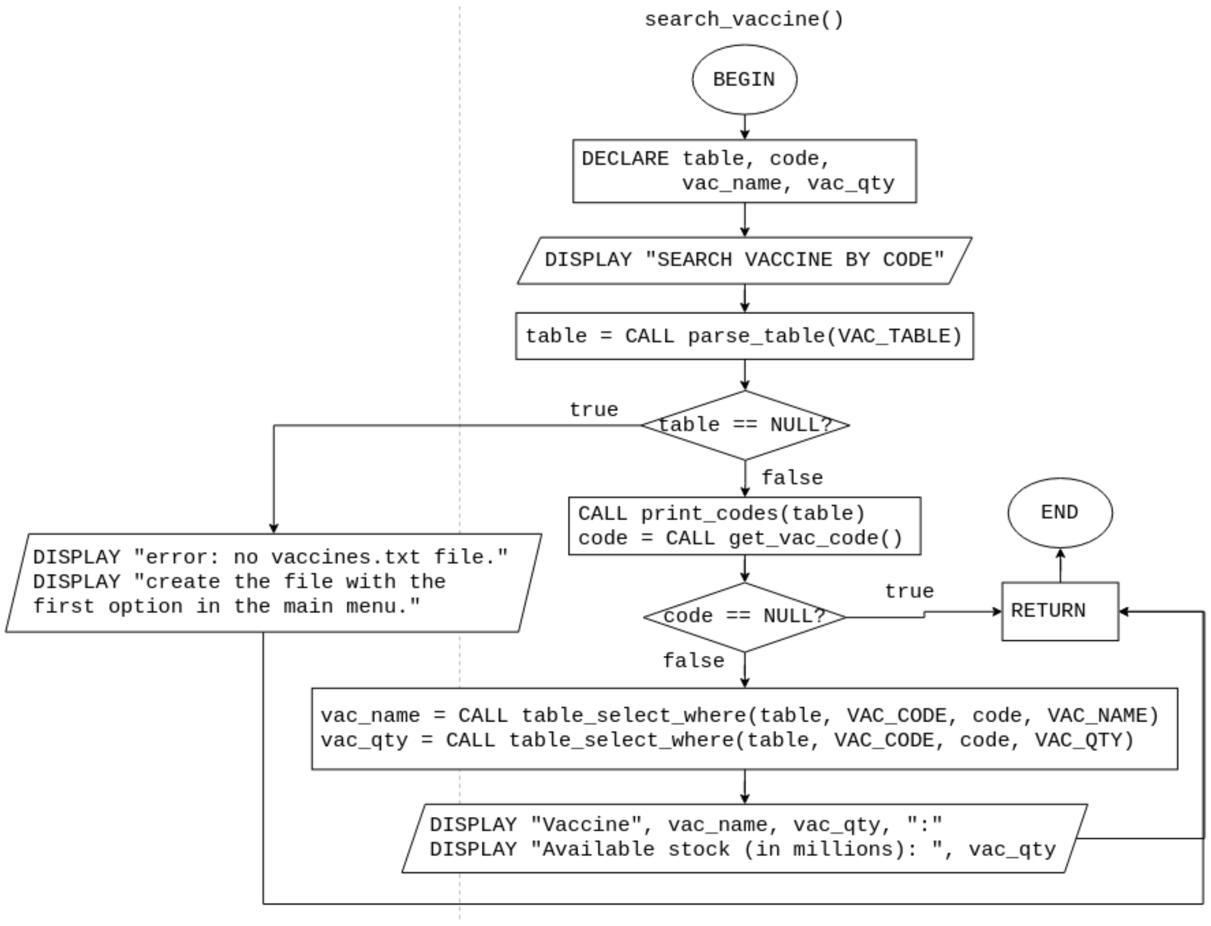
9. write_dist function

```
FUNCTION write_dist(code, amt)
BEGIN
  DECLARE fp, datetime
  fp = OPEN "dist.txt" for reading
  IF fp == NULL THEN
    CALL create_dist_txt()
  ELSE
    CLOSE fp
  ENDIF
  fp = OPEN "dist.txt" for appending
  datetime = CALL get_current_datetime()
  WRITE fp code, amt, datetime
  CLOSE fp
END
```


write_dist(code, amt)
BEGIN
DECLARE fp, datetime
fp = OPEN "dist.txt" for reading
fp == NULL?
false
true
CLOSE fp
CALL create_dist_txt()
fp = OPEN "dist.txt" for appending
datetime = CALL get_current_datetime()
WRITE fp code, amt, datetime
CLOSE fp
RETURN
END

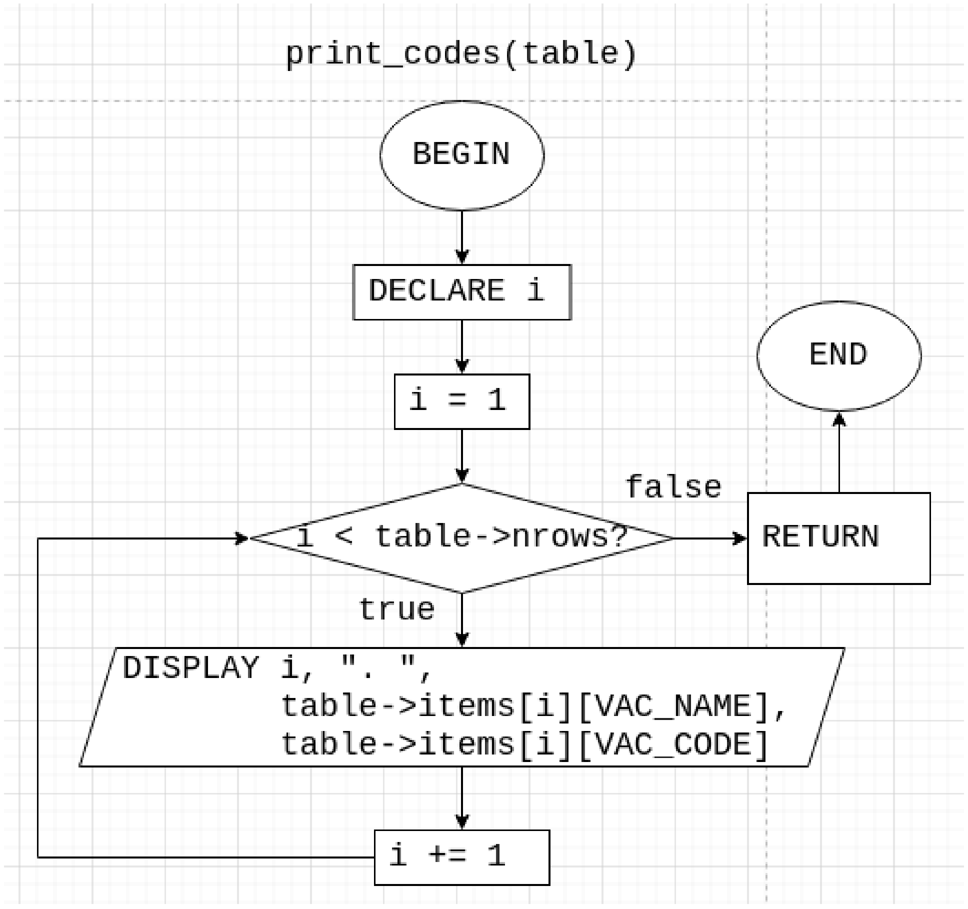
10. search_vaccine function

```
FUNCTION search_vaccine()
BEGIN
  DECLARE table, code, vac_name, vac_qty
  DISPLAY "SEARCH VACCINE BY CODE"
  table = CALL parse_table(VAC_TABLE)
  IF table == NULL THEN
    DISPLAY "error: no vaccines.txt file."
    DISPLAY "create the file with the first option in the main menu."
    RETURN
  ENDIF
  CALL print_codes(table)
  code = CALL get_vac_code()
  IF code == NULL THEN
    RETURN
  ENDIF
  vac_name = CALL table_select_where(table, VAC_CODE, code, VAC_NAME)
  vac_qty = CALL table_select_where(table, VAC_CODE, code, VAC_QTY)
  DISPLAY "Vaccine", vac_name, vac_qty, ":"
  DISPLAY "Available stock (in millions): ", vac_qty
END
```

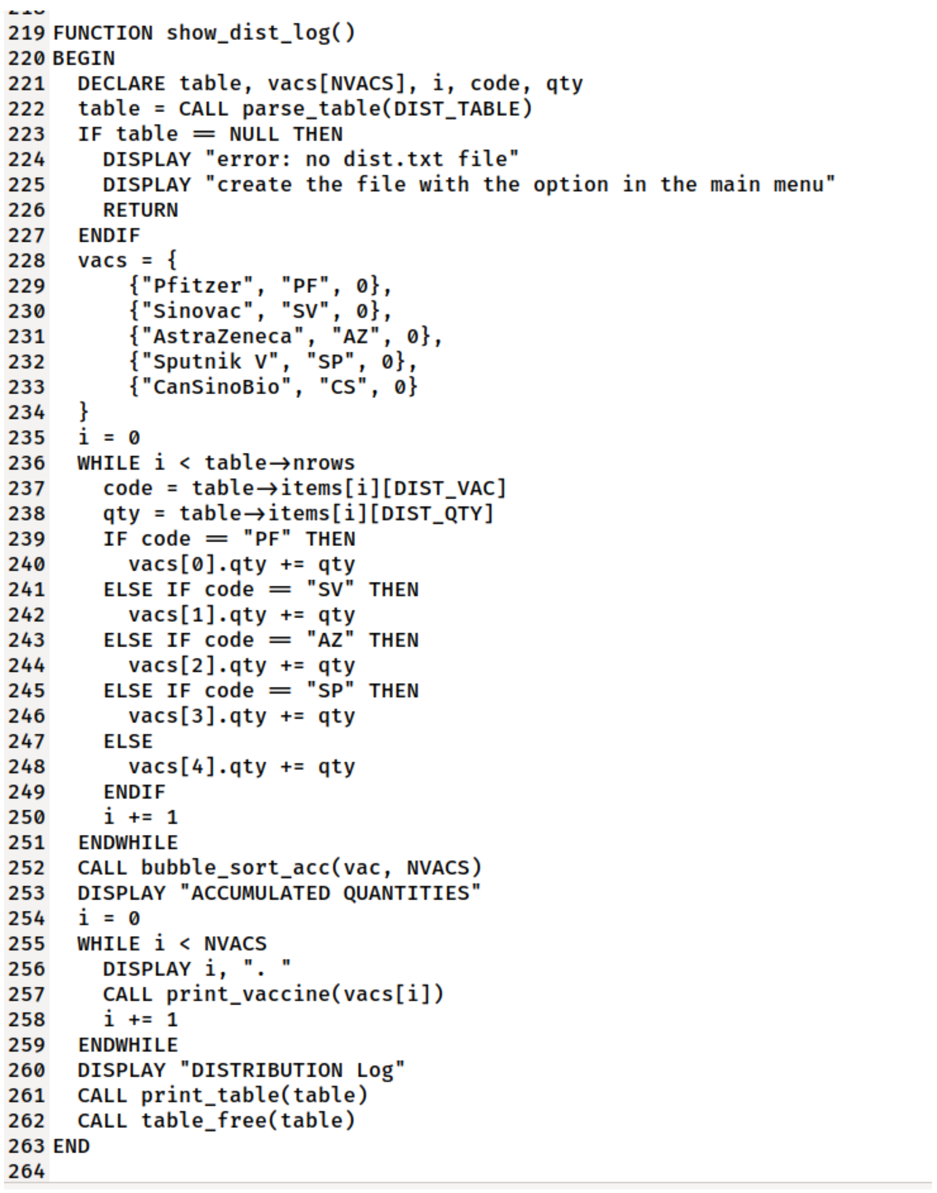

search_vaccine()
BEGIN
DECLARE table, code, vac_name, vac_qty
DISPLAY "SEARCH VACCINE BY CODE"
table = CALL parse_table(VAC_TABLE)
table == NULL?
true
false
DISPLAY "error: no vaccines.txt file." DISPLAY "create the file with the first option in the main menu."
CALL print_codes(table) code = CALL get_vac_code()
code == NULL?
true
false
RETURN
END
vac_name = CALL table_select_where(table, VAC_CODE, code, VAC_NAME) vac_qty = CALL table_select_where(table, VAC_CODE, code, VAC_QTY)
DISPLAY "Vaccine", vac_name, vac_qty, ":" DISPLAY "Available stock (in millions): ", vac_qty

11. print_codes function

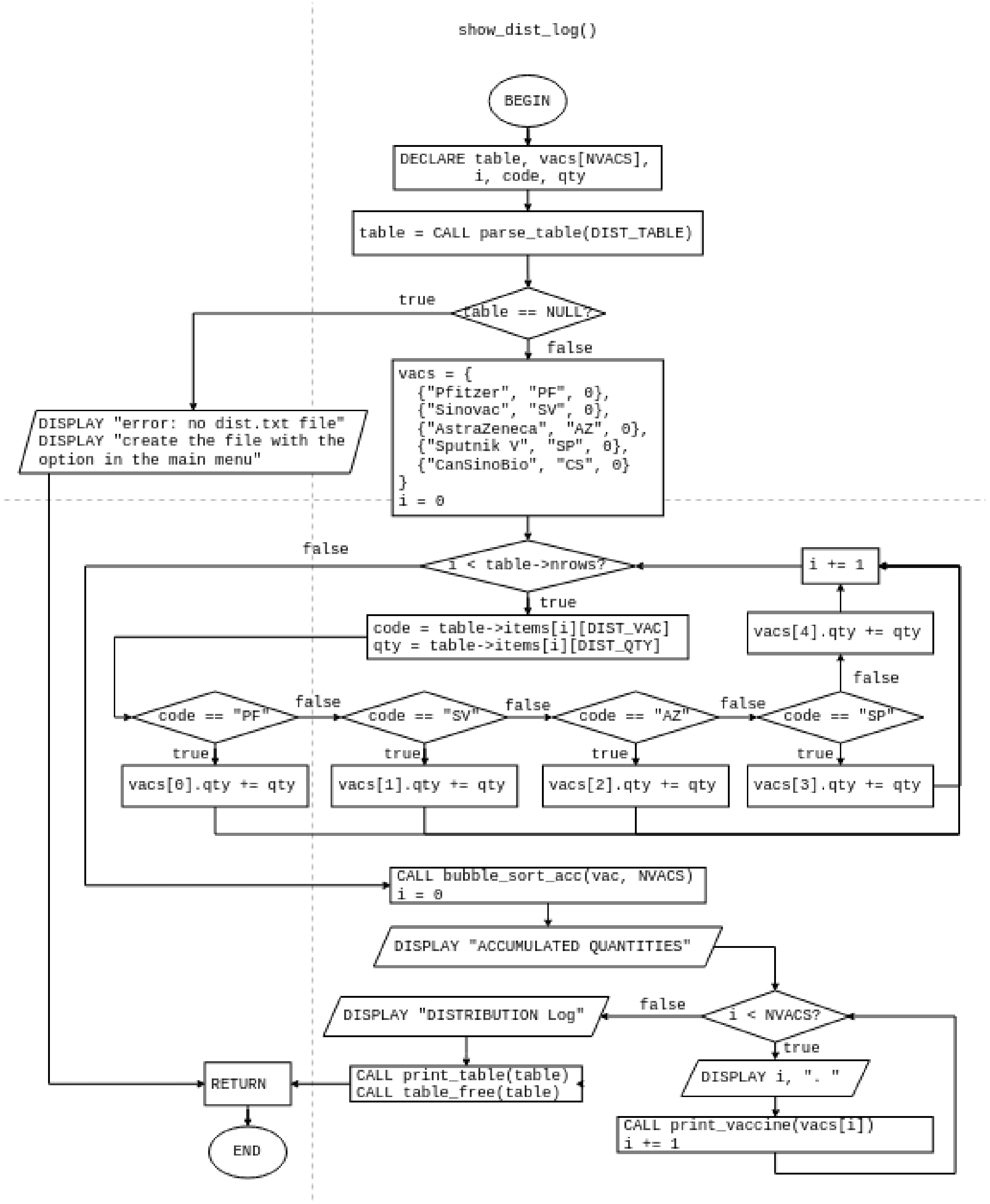
```
FUNCTION print_codes
BEGIN
  DECLARE i
  i = 1
  WHILE i < table→nrows
    DISPLAY i, ". ", table→items[i][VAC_NAME], table→items[i][VAC_CODE]
    i += 1
  ENDWHILE
END
```



12. show_dist_log function

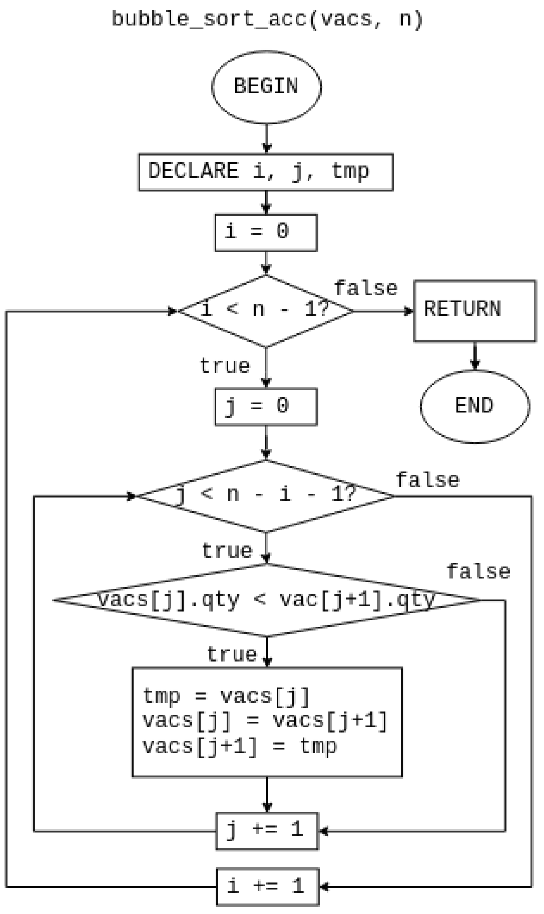


```
FUNCTION show_dist_log()
BEGIN
  DECLARE table, vacs[NVACS], i, code, qty
  table = CALL parse_table(DIST_TABLE)
  IF table == NULL THEN
    DISPLAY "error: no dist.txt file"
    DISPLAY "create the file with the option in the main menu"
    RETURN
  ENDIF
  vacs = {
      {"Pfitzer", "PF", 0},
      {"Sinovac", "SV", 0},
      {"AstraZeneca", "AZ", 0},
      {"Sputnik V", "SP", 0},
      {"CanSinoBio", "CS", 0}
  }
  i = 0
  WHILE i < table->nrows
    code = table->items[i][DIST_VAC]
    qty = table->items[i][DIST_QTY]
    IF code == "PF" THEN
      vacs[0].qty += qty
    ELSE IF code == "SV" THEN
      vacs[1].qty += qty
    ELSE IF code == "AZ" THEN
      vacs[2].qty += qty
    ELSE IF code == "SP" THEN
      vacs[3].qty += qty
    ELSE
      vacs[4].qty += qty
    ENDIF
    i += 1
  ENDWHILE
  CALL bubble_sort_acc(vac, NVACS)
  DISPLAY "ACCUMULATED QUANTITIES"
  i = 0
  WHILE i < NVACS
    DISPLAY i, ". "
    CALL print_vaccine(vacs[i])
    i += 1
  ENDWHILE
  DISPLAY "DISTRIBUTION Log"
  CALL print_table(table)
  CALL table_free(table)
END
```


show_dist_log()
BEGIN
DECLARE table, vacs[NVACS], i, code, qty
table = CALL parse_table(DIST_TABLE)
table == NULL?
true
false
DISPLAY "error: no dist.txt file"
DISPLAY "create the file with the option in the main menu"
vacs = {
{"Pfitzer", "PF", 0},
{"Sinovac", "SV", 0},
{"AstraZeneca", "AZ", 0},
{"Sputnik V", "SP", 0},
{"CanSinoBio", "CS", 0}
}
i = 0
i < table->nrows?
false
true
i += 1
code = table->items[i][DIST_VAC]
qty = table->items[i][DIST_QTY]
vacs[4].qty += qty
code == "PF"
code == "SV"
code == "AZ"
code == "SP"
false
true
vacs[0].qty += qty
vacs[1].qty += qty
vacs[2].qty += qty
vacs[3].qty += qty
CALL bubble_sort_acc(vac, NVACS)
i = 0
DISPLAY "ACCUMULATED QUANTITIES"
i < NVACS?
false
true
DISPLAY "DISTRIBUTION Log"
DISPLAY i, ". "
CALL print_vaccine(vacs[i])
i += 1
CALL print_table(table)
CALL table_free(table)
RETURN
END

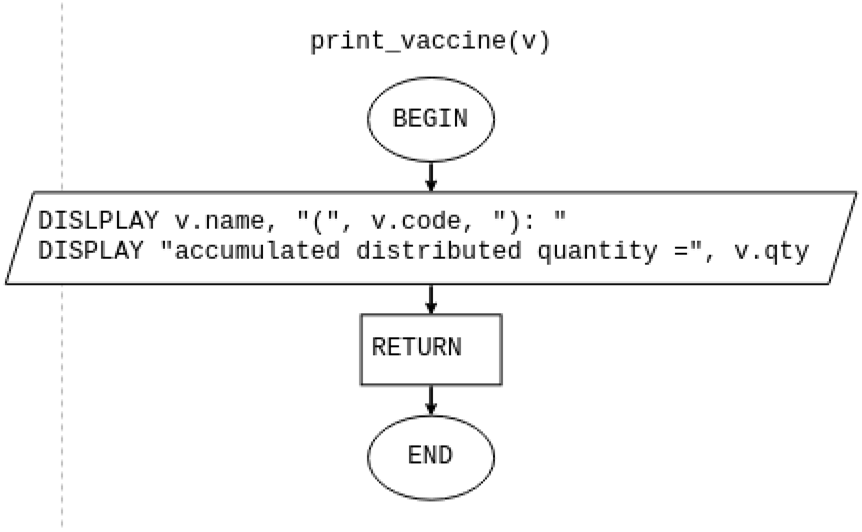
13. bubble_sort_acc function

```
FUNCTION bubble_sort_acc(vacs, n)
BEGIN
  DECLARE i, j, tmp
  i = 0
  WHILE i < n - 1
    j = 0
    WHILE j < n - i - 1
      IF vacs[j].qty < vac[j+1].qty THEN
        tmp = vacs[j]
        vacs[j] = vacs[j+1]
        vacs[j+1] = tmp
      ENDIF
      j += 1
    ENDWHILE
    i += 1
  ENDWHILE
END
```



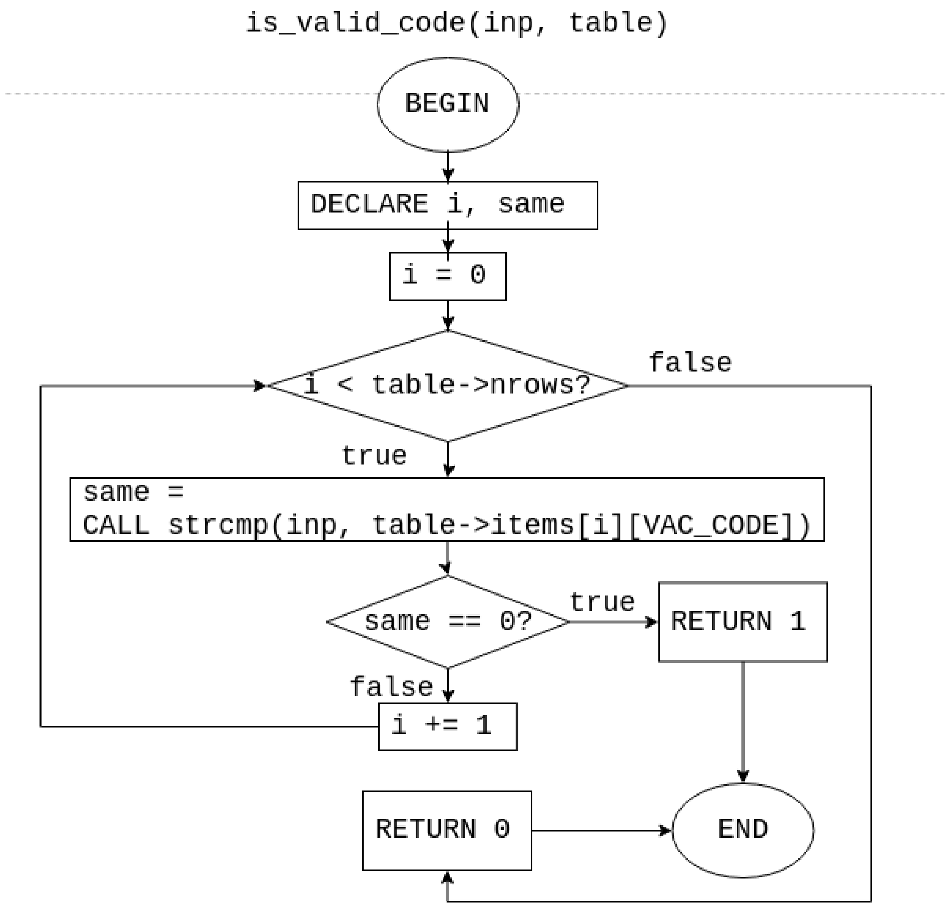
14. print_vaccine function

```
FUNCTION print_vaccine(v)
BEGIN
  DISLPLAY v.name, "(", v.code, "): "
  DISPLAY "accumulated distributed quantity =", v.qty
END

```


print_vaccine(v)
BEGIN
DISLPLAY v.name, "(", v.code, "): "
DISPLAY "accumulated distributed quantity =", v.qty
RETURN
END

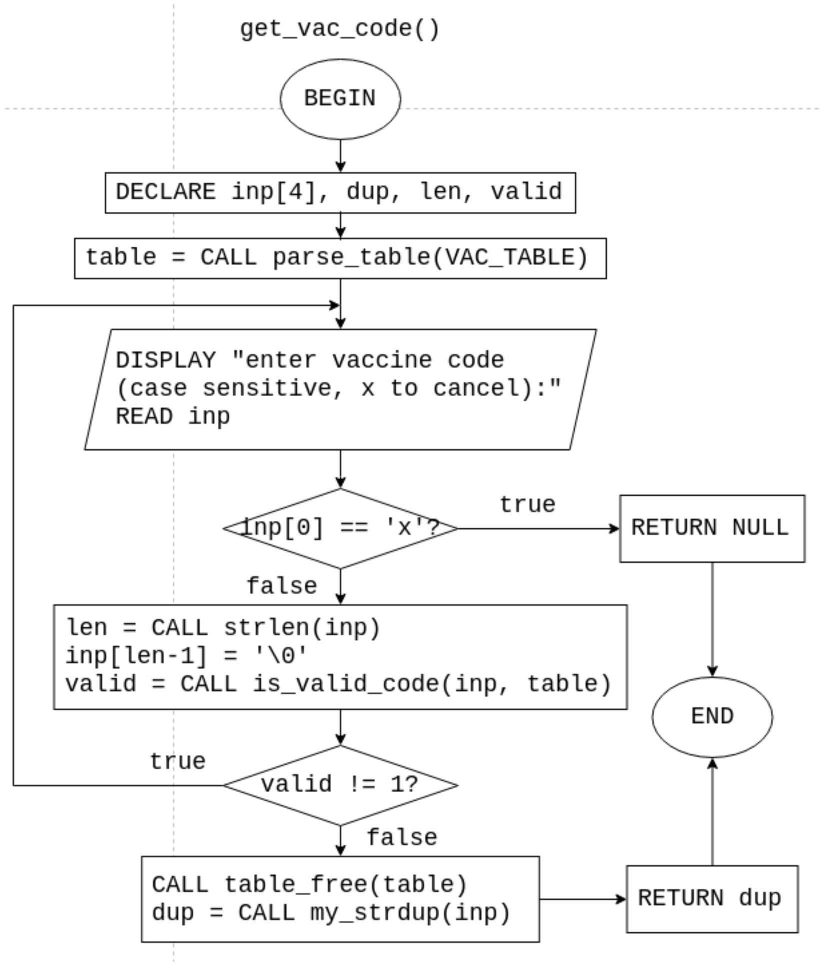
15. is_valid_code function

```
FUNCTION is_valid_code(inp, table)
BEGIN
  DECLARE i, same
  i = 0
  WHILE i < table→nrows
    same = CALL strcmp(inp, table→items[i][VAC_CODE])
    IF same == 0 THEN
      RETURN 1
    ENDIF
    i += 1
  ENDWHILE
  RETURN 0
END
```

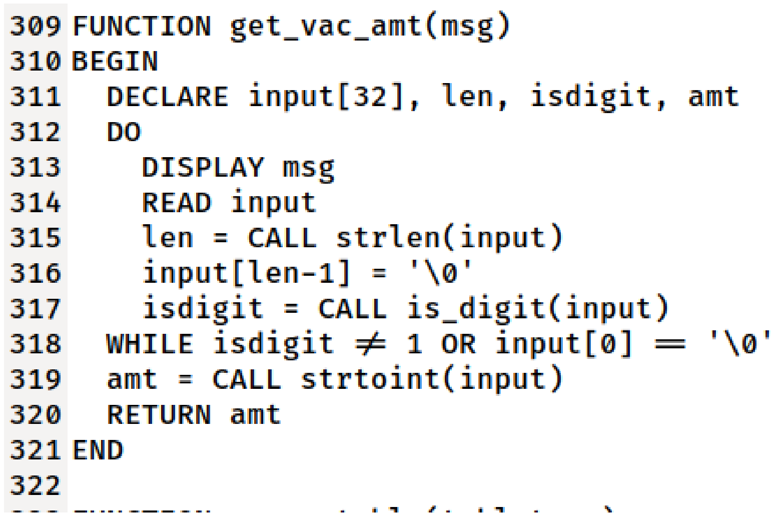


16. get_vac_code function

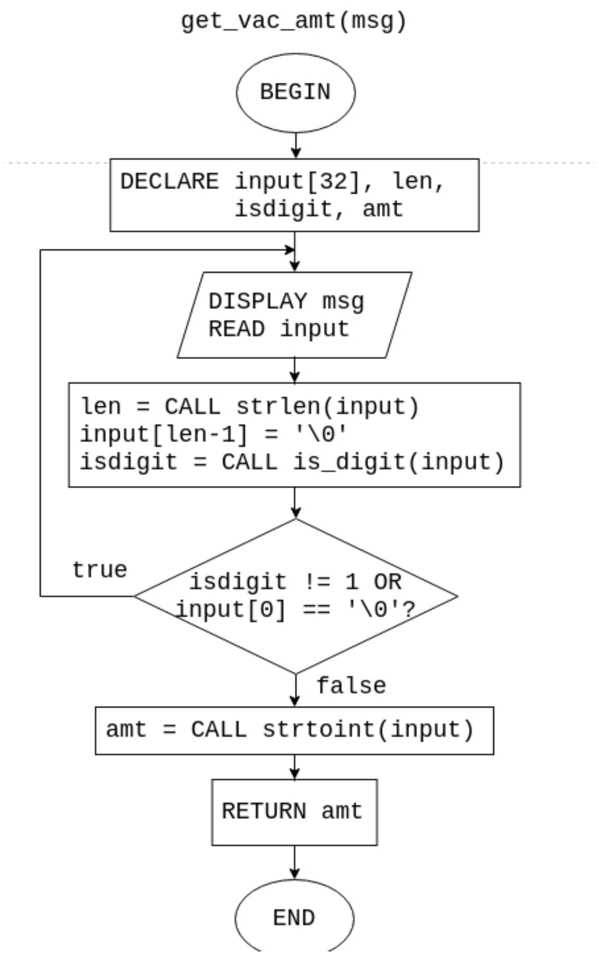
```
FUNCTION get_vac_code()
BEGIN
  DECLARE inp[4], dup, len, valid
  table = CALL parse_table(VAC_TABLE)
  DO
    DISPLAY "enter vaccine code (case sensitive, x to cancel): "
    READ inp
    IF inp[0] == 'x' THEN
      RETURN NULL
    ENDIF
    len = CALL strlen(inp)
    inp[len-1] = '\0'
    valid = CALL is_valid_code(inp, table)
  WHILE valid ≠ 1
  CALL table_free(table)
  dup = CALL my_strdup(inp)
  RETURN dup
END
```


get_vac_code()
BEGIN
DECLARE inp[4], dup, len, valid
table = CALL parse_table(VAC_TABLE)
DISPLAY "enter vaccine code (case sensitive, x to cancel):" READ inp
inp[0] == 'x'?
true
RETURN NULL
false
len = CALL strlen(inp) inp[len-1] = '\0' valid = CALL is_valid_code(inp, table)
END
true
valid != 1?
false
CALL table_free(table) dup = CALL my_strdup(inp)
RETURN dup

17. get_vac_amt function

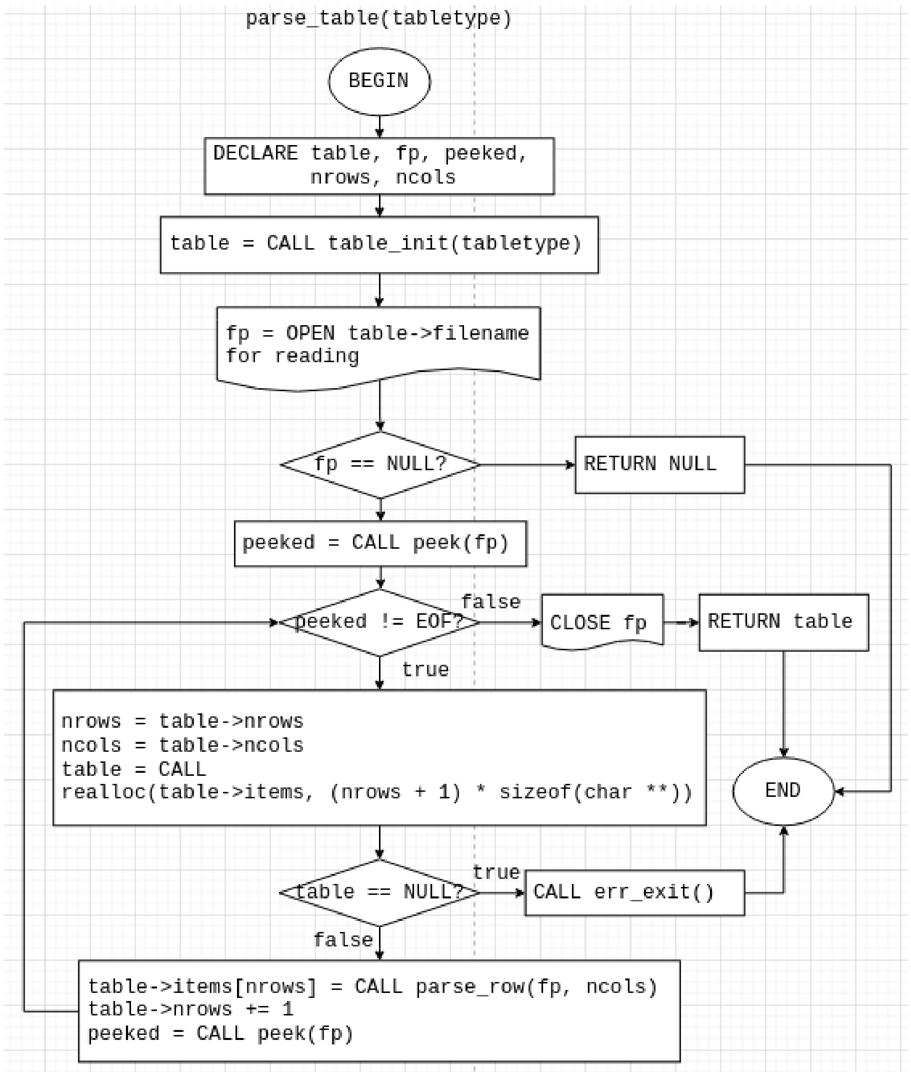


```
FUNCTION get_vac_amt(msg)
BEGIN
  DECLARE input[32], len, isdigit, amt
  DO
    DISPLAY msg
    READ input
    len = CALL strlen(input)
    input[len-1] = '\0'
    isdigit = CALL is_digit(input)
  WHILE isdigit ≠ 1 OR input[0] == '\0'
  amt = CALL strtoint(input)
  RETURN amt
END
```


get_vac_amt(msg)
BEGIN
DECLARE input[32], len, isdigit, amt
DISPLAY msg READ input
len = CALL strlen(input) input[len-1] = '\0' isdigit = CALL is_digit(input)
isdigit != 1 OR input[0] == '\0'?
true
false
amt = CALL strtoint(input)
RETURN amt
END

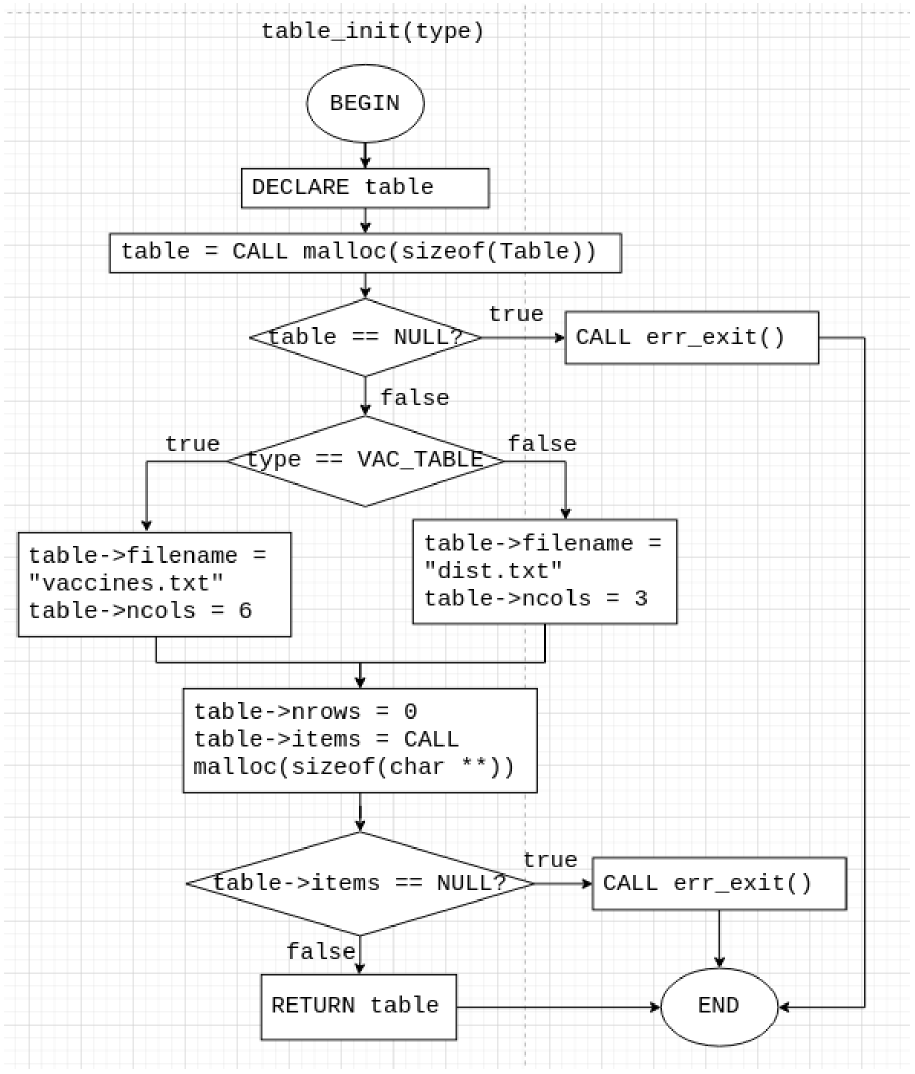
18. parse_table function

```
FUNCTION parse_table(tabletype)
BEGIN
  DECLARE table, fp, peeked, nrows, ncols
  table = CALL table_init(tabletype)
  fp = OPEN table→filename for reading
  IF fp == NULL THEN
    RETURN NULL
  ENDIF
  peeked = CALL peek(fp)
  WHILE peeked ≠ EOF
    nrows = table→nrows
    ncols = table→ncols
    table = CALL realloc(table→items, (nrows + 1) * sizeof(char **))
    IF table == NULL THEN
      CALL err_exit()
    ENDIF
    table→items[nrows] = CALL parse_row(fp, ncols)
    table→nrows += 1
    peeked = CALL peek(fp)
  ENDWHILE
  CLOSE fp
  RETURN table
END
```

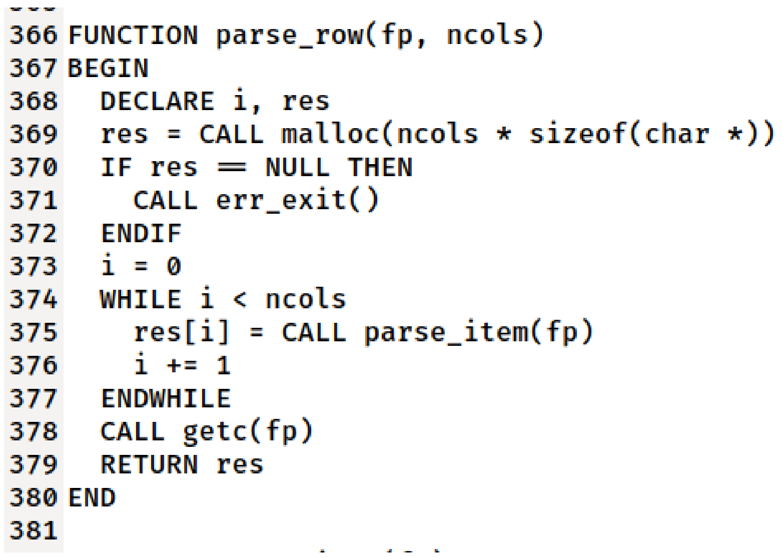


19. table_init function

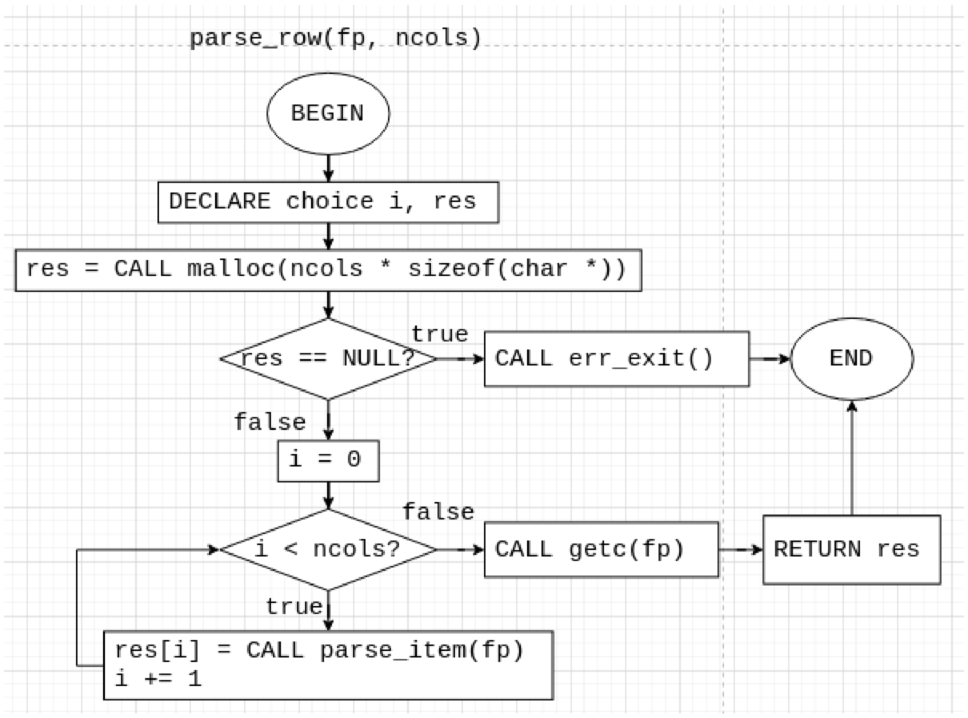
```
FUNCTION table_init(type)
BEGIN
  DECLARE table
  table = CALL malloc(sizeof(Table))
  IF table == NULL THEN
    CALL err_exit()
  ENDIF
  IF type == VAC_TABLE THEN
    table→filename = "vaccines.txt"
    table→ncols = 6
  ELSE
    table→filename = "dist.txt"
    table→ncols = 3
  ENDIF
  table→nrows = 0
  table→items = CALL malloc(sizeof(char **))
  IF table→items == NULL THEN
    CALL err_exit()
  ENDIF
  RETURN table
END
```


table_init(type)
BEGIN
DECLARE table
table = CALL malloc(sizeof(Table))
table == NULL?
true
CALL err_exit()
false
type == VAC_TABLE
true
false
table->filename = "vaccines.txt" table->ncols = 6
table->filename = "dist.txt" table->ncols = 3
table->nrows = 0 table->items = CALL malloc(sizeof(char **))
table->items == NULL?
true
CALL err_exit()
false
RETURN table
END

20. parse_row function

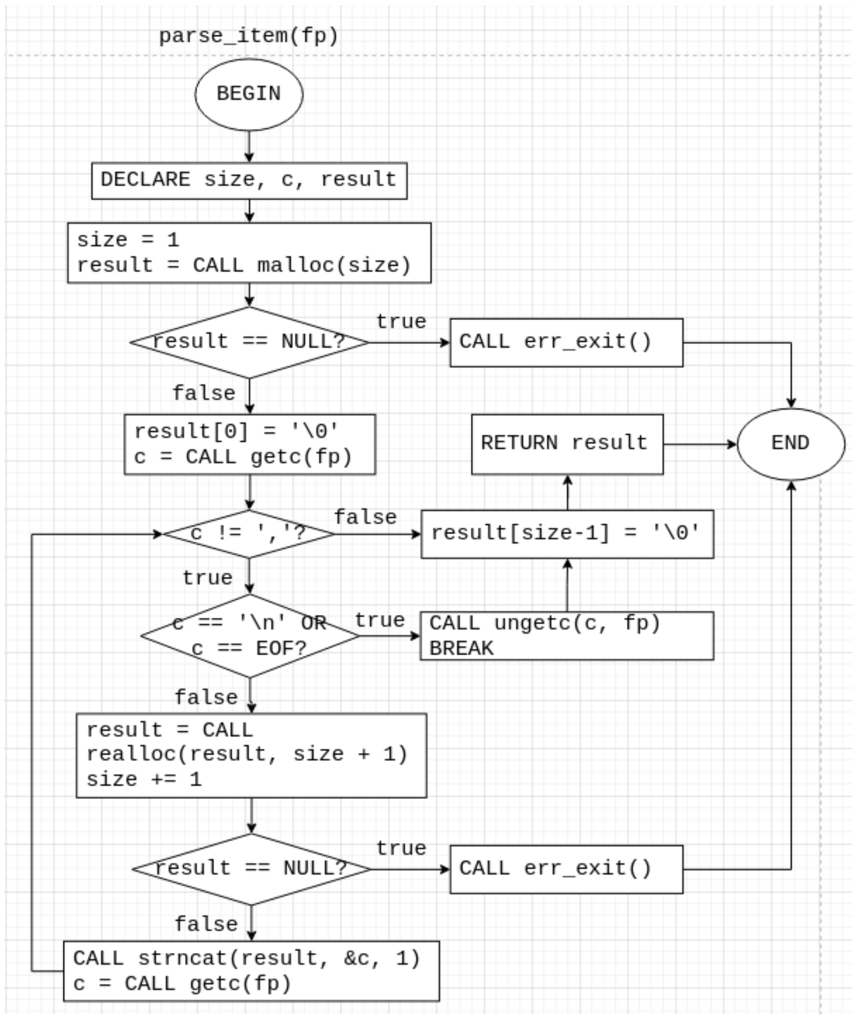


```
FUNCTION parse_row(fp, ncols)
BEGIN
  DECLARE i, res
  res = CALL malloc(ncols * sizeof(char *))
  IF res == NULL THEN
    CALL err_exit()
  ENDIF
  i = 0
  WHILE i < ncols
    res[i] = CALL parse_item(fp)
    i += 1
  ENDWHILE
  CALL getc(fp)
  RETURN res
END
```



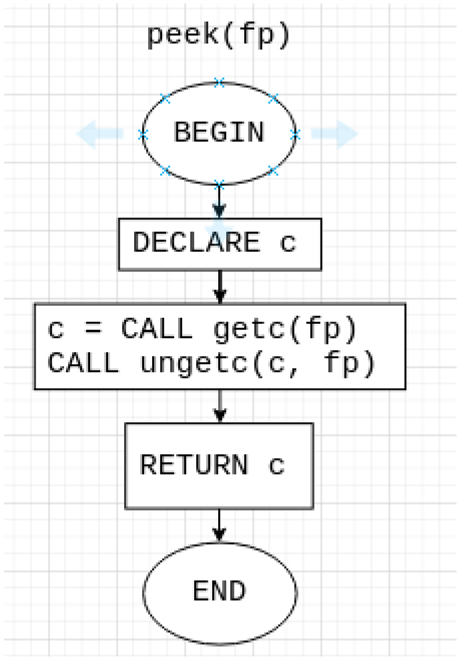
21. parse_item function

```
FUNCTION parse_item(fp)
BEGIN
  DECLARE size, c, result
  size = 1
  result = CALL malloc(size)
  IF result == NULL THEN
    CALL err_exit()
  ENDIF
  result[0] = '\0'
  c = CALL getc(fp)
  WHILE c != ','
    IF c == '\n' OR c == EOF THEN
      CALL ungetc(c, fp)
      BREAK
    ENDIF
    result = CALL realloc(result, size + 1)
    size += 1
    IF result == NULL THEN
      CALL err_exit()
    ENDIF
    CALL strncat(result, &c, 1)
    c = CALL getc(fp)
  ENDWHILE
  result[size-1] = '\0'
  RETURN result
END
```



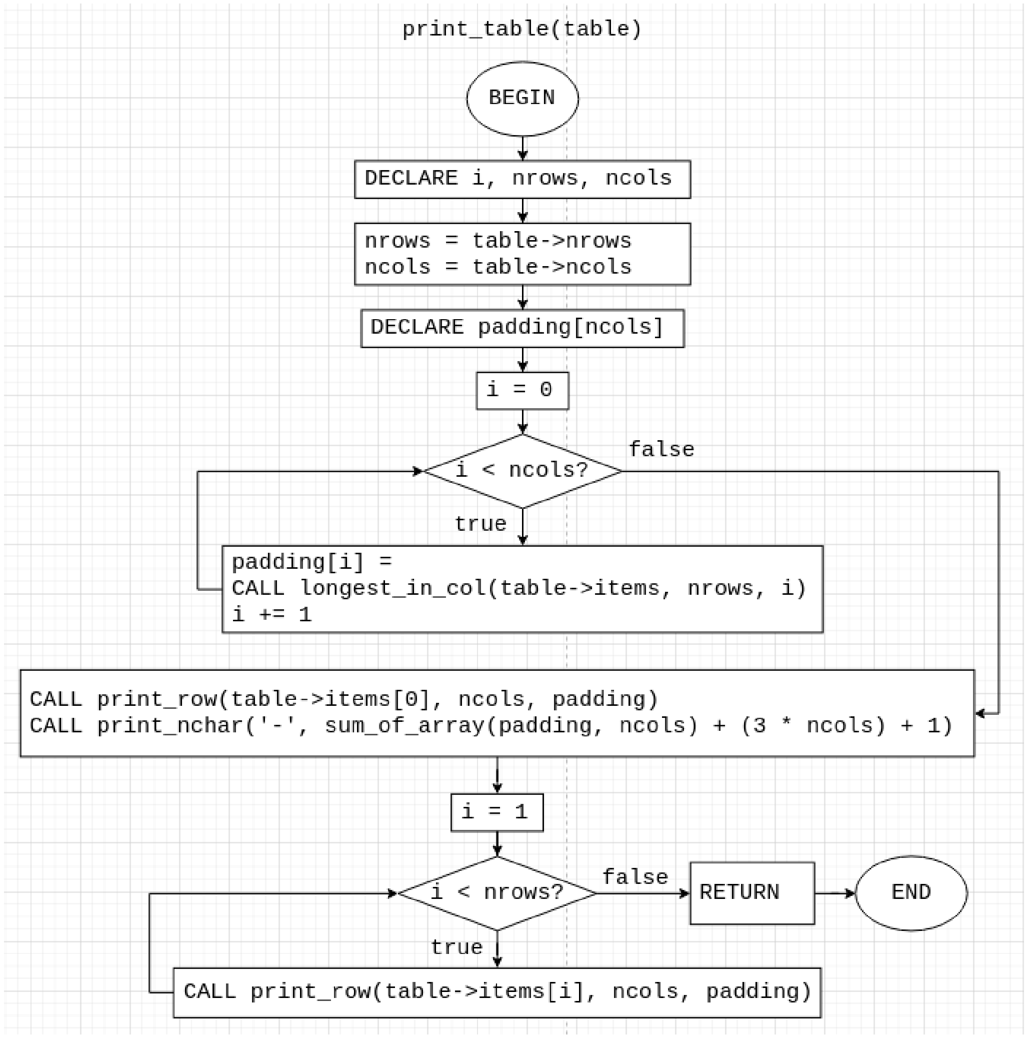
22. peek function

```
FUNCTION peek(fp)
BEGIN
  DECLARE c
  c = CALL getc(fp)
  CALL ungetc(c, fp)
  RETURN c
END
```

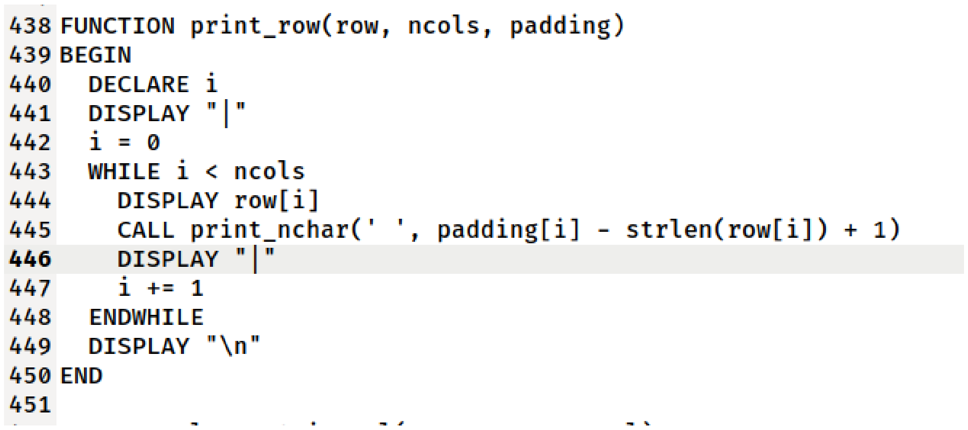


23. print_table function

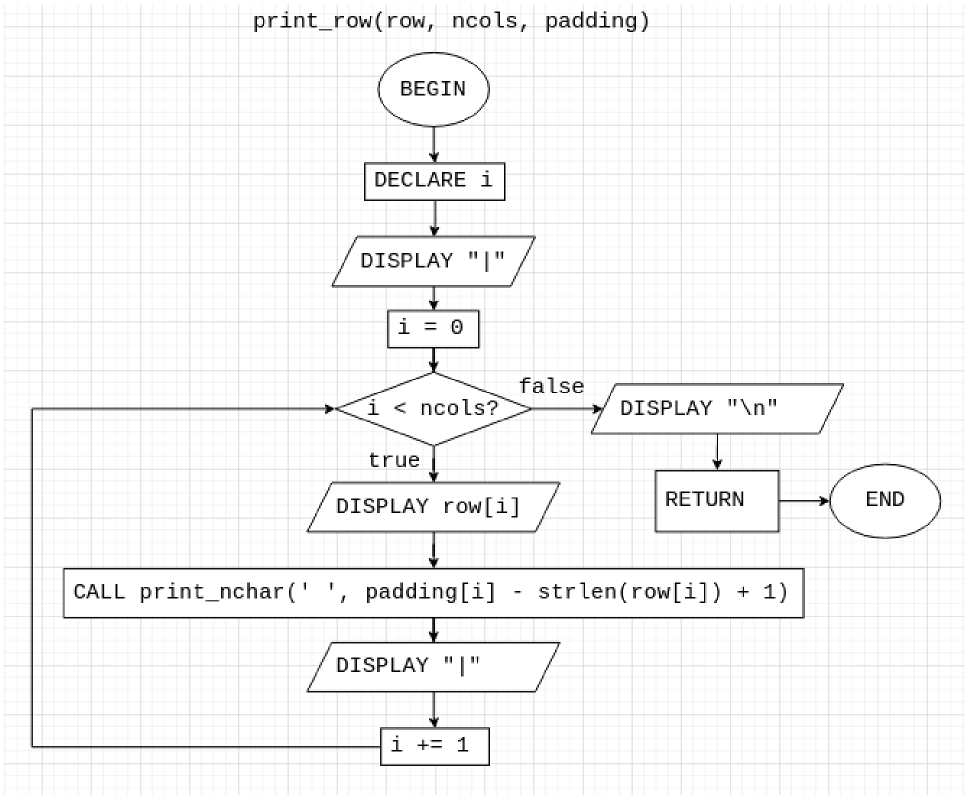
```
FUNCTION print_table(table)
BEGIN
  DECLARE i, nrows, ncols
  nrows = table→nrows
  ncols = table→ncols
  DECLARE  padding[ncols]
  i = 0
  WHILE i < ncols
    padding[i] = CALL longest_in_col(table→items, nrows, i)
    i += 1
  ENDWHILE
  CALL print_row(table→items[0], ncols, padding)
  CALL print_nchar('-', sum_of_array(padding, ncols) + (3 * ncols) + 1)
  i = 1
  WHILE i < nrows
    CALL print_row(table→items[i], ncols, padding)
  ENDWHILE
END
```


print_table(table)
BEGIN
DECLARE i, nrows, ncols
nrows = table->nrows
ncols = table->ncols
DECLARE padding[ncols]
i = 0
i < ncols?
false
true
padding[i] =
CALL longest_in_col(table->items, nrows, i)
i += 1
CALL print_row(table->items[0], ncols, padding)
CALL print_nchar('-', sum_of_array(padding, ncols) + (3 * ncols) + 1)
i = 1
i < nrows?
false
RETURN
END
true
CALL print_row(table->items[i], ncols, padding)

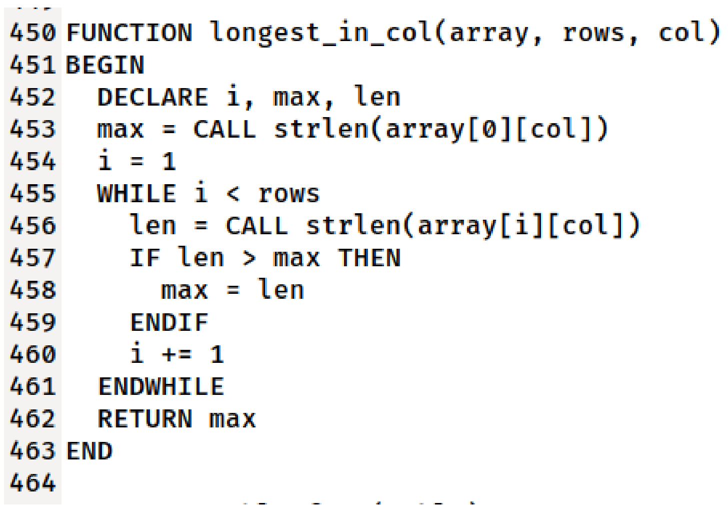
24. print_row function



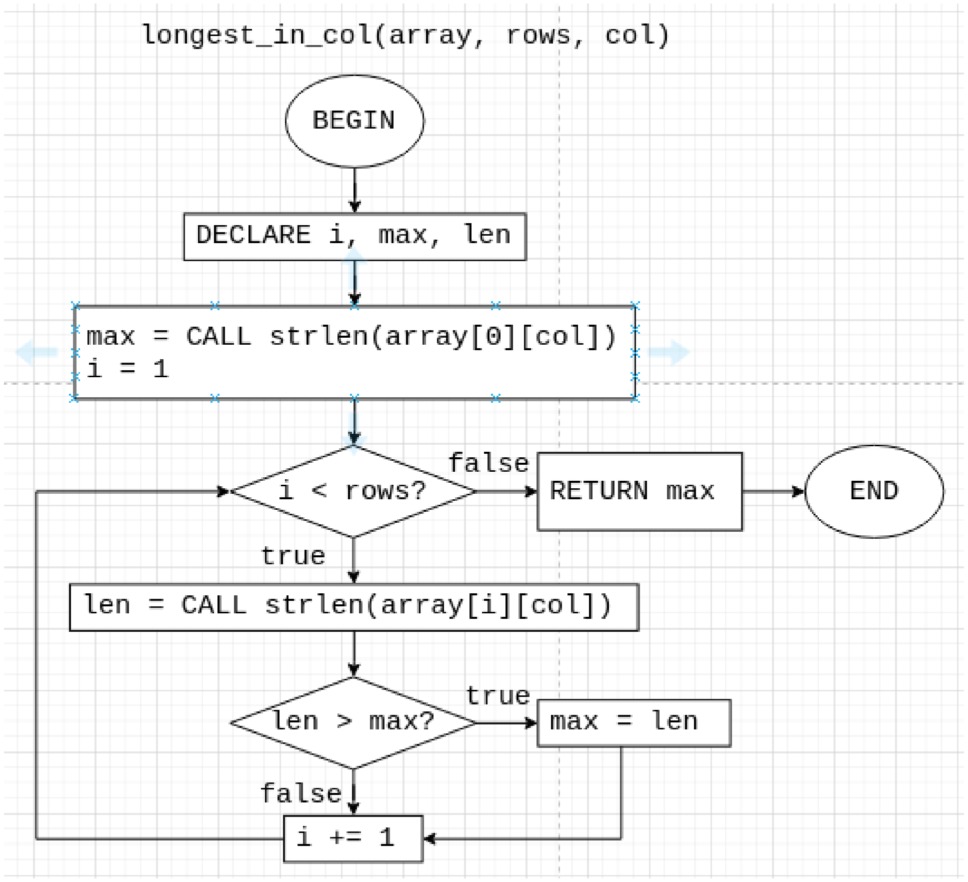
```
FUNCTION print_row(row, ncols, padding)
BEGIN
  DECLARE i
  DISPLAY "|"
  i = 0
  WHILE i < ncols
    DISPLAY row[i]
    CALL print_nchar(' ', padding[i] - strlen(row[i]) + 1)
    DISPLAY "|"
    i += 1
  ENDWHILE
  DISPLAY "\n"
END
```



25. longest_in_col function

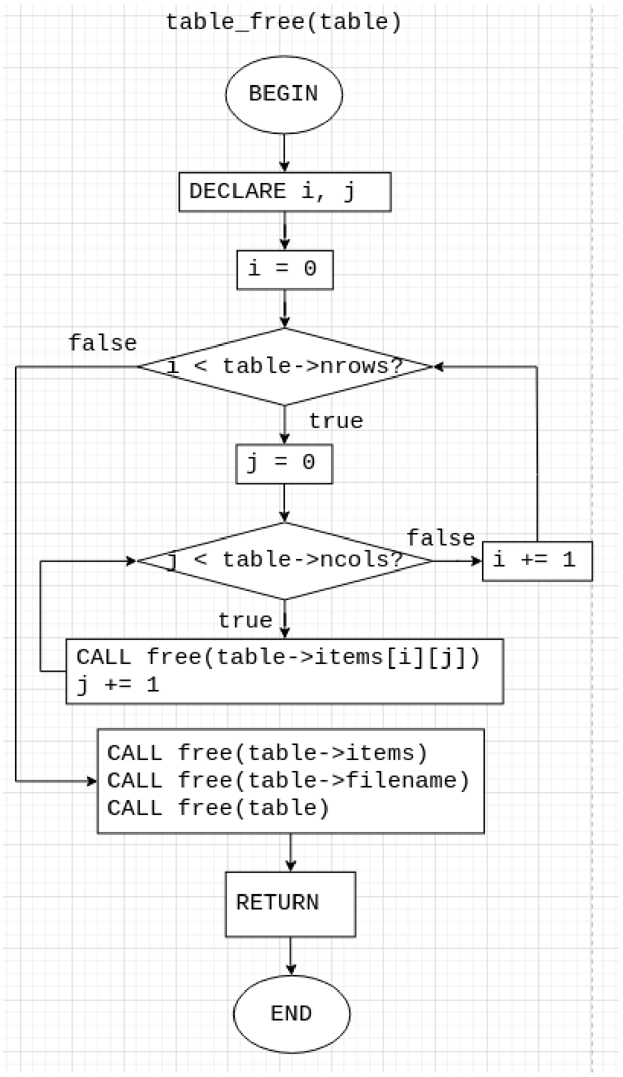


```
FUNCTION longest_in_col(array, rows, col)
BEGIN
  DECLARE i, max, len
  max = CALL strlen(array[0][col])
  i = 1
  WHILE i < rows
    len = CALL strlen(array[i][col])
    IF len > max THEN
      max = len
    ENDIF
    i += 1
  ENDWHILE
  RETURN max
END
```


longest_in_col(array, rows, col)
BEGIN
DECLARE i, max, len
max = CALL strlen(array[0][col])
i = 1
i < rows?
false
RETURN max
END
true
len = CALL strlen(array[i][col])
len > max?
true
max = len
false
i += 1

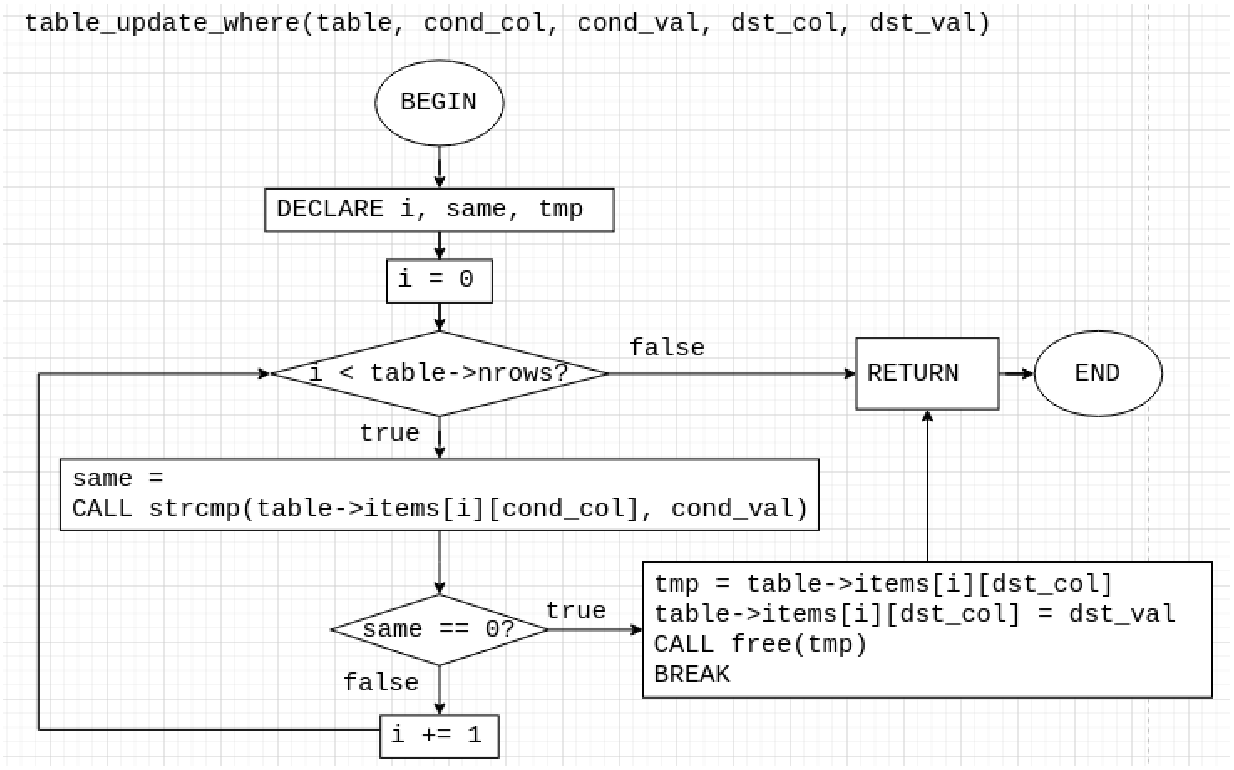
26. table_free function

```
FUNCTION table_free(table)
BEGIN
  DECLARE i, j
  i = 0
  WHILE i < table→nrows
    j = 0
    WHILE j < table→ncols
      CALL free(table→items[i][j])
      j += 1
    ENDWHILE
    CALL free(table→items[i])
    i += 1
  ENDWHILE
  CALL free(table→items)
  CALL free(table→filename)
  CALL free(table)
END
```



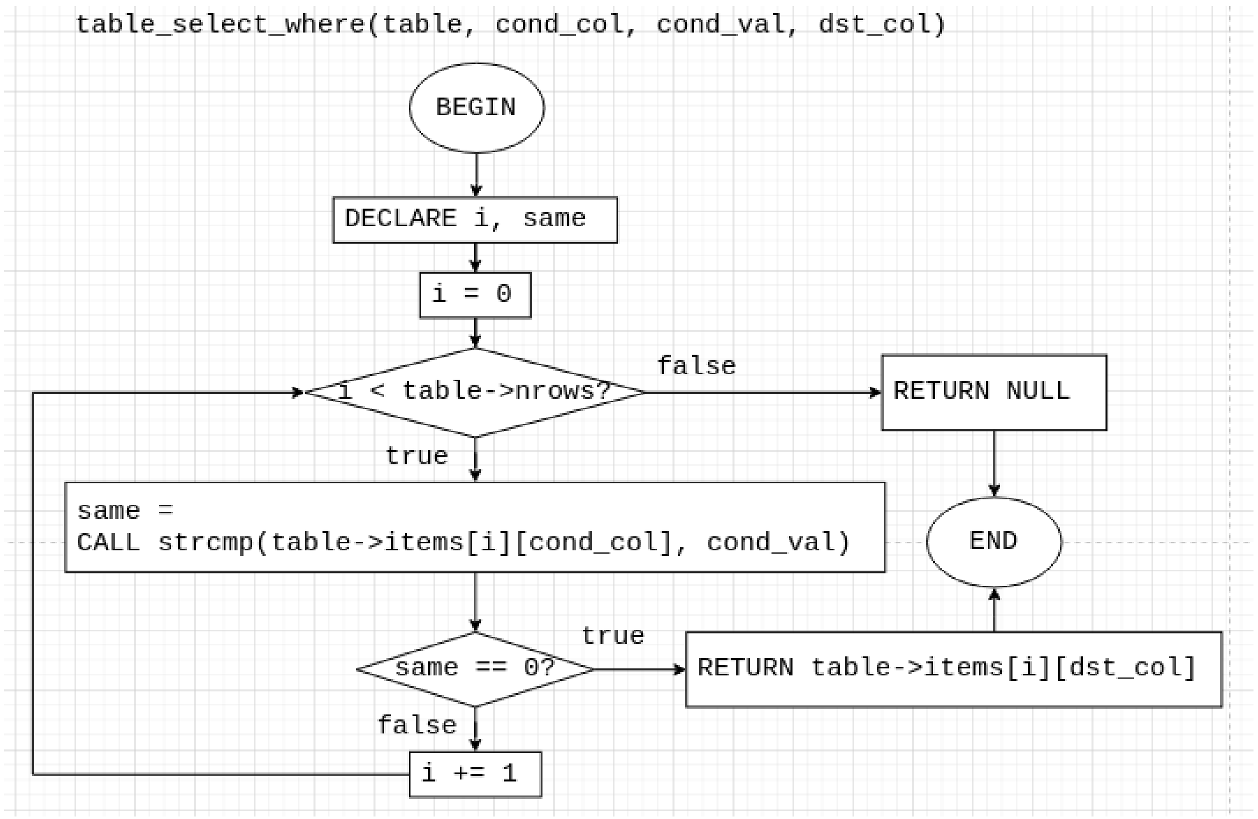
27. table_update_where function

```
FUNCTION table_update_where(table, cond_col, cond_val, dst_col, dst_val)
BEGIN
  DECLARE i, same, tmp
  i = 0
  WHILE i < table→nrows
    same = CALL strcmp(table→items[i][cond_col], cond_val)
    IF same == 0 THEN
      tmp = table→items[i][dst_col]
      table→items[i][dst_col] = dst_val
      CALL free(tmp)
      BREAK
    ENDIF
    i += 1
  ENDWHILE
END
```


table_update_where(table, cond_col, cond_val, dst_col, dst_val)
BEGIN
DECLARE i, same, tmp
i = 0
i < table->nrows?
false
RETURN
END
true
same =
CALL strcmp(table->items[i][cond_col], cond_val)
same == 0?
true
tmp = table->items[i][dst_col]
table->items[i][dst_col] = dst_val
CALL free(tmp)
BREAK
false
i += 1

28. table_select_where function

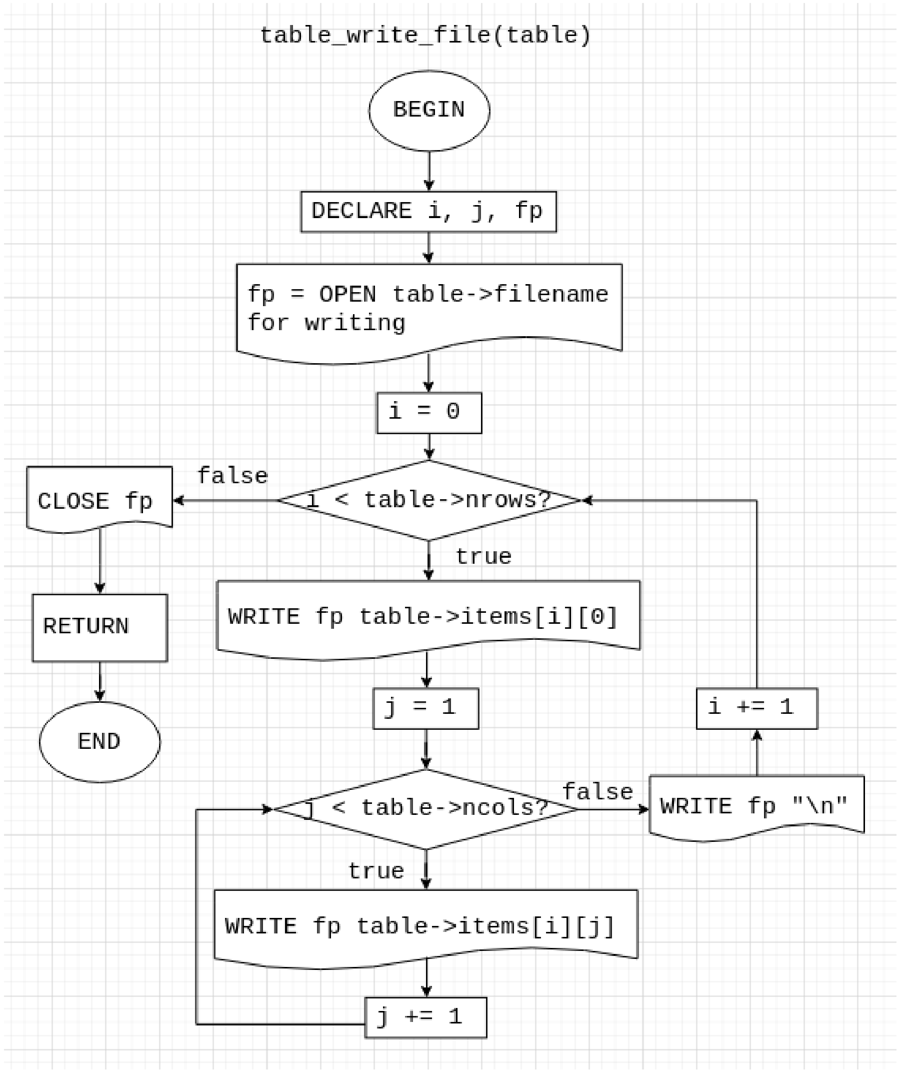
```
FUNCTION table_select_where(table, cond_col, cond_val, dst_col)
BEGIN
  DECLARE i, same
  i = 0
  WHILE i < table->nrows
    same = CALL strcmp(table->items[i][cond_col], cond_val)
    IF same == 0 THEN
      RETURN table->items[i][dst_col]
    ENDIF
    i += 1
  ENDWHILE
  RETURN NULL
END
```



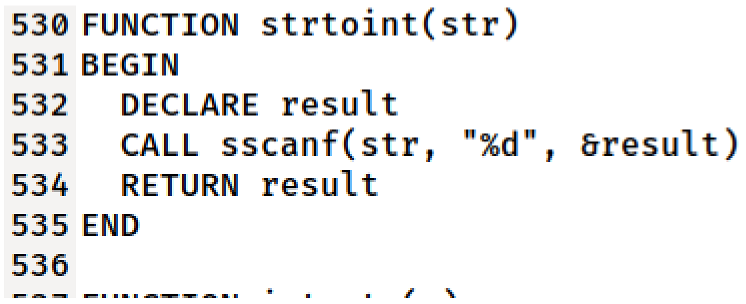
table_select_where(table, cond_col, cond_val, dst_col)
BEGIN
DECLARE i, same
i = 0
i < table->nrows?
false
RETURN NULL
true
same =
CALL strcmp(table->items[i][cond_col], cond_val)
END
true
same == 0?
RETURN table->items[i][dst_col]
false
i += 1

29. table_write_file function

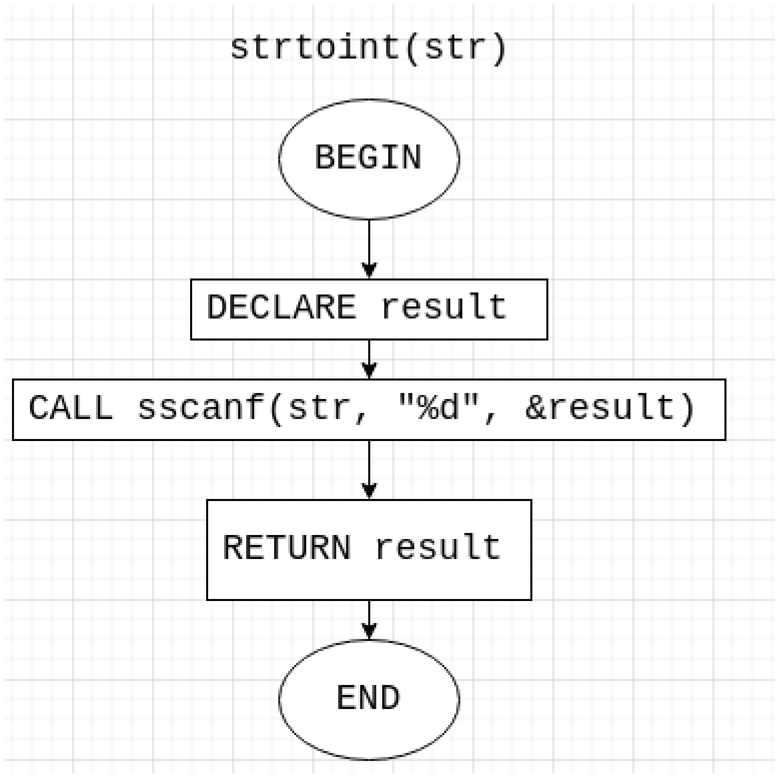
```
FUNCTION table_write_file(table)
BEGIN
  DECLARE i, j, fp
  fp = OPEN table→filename for writing
  i = 0
  WHILE i < table→nrows
    WRITE fp table→items[i][0]
    j = 1
    WHILE j < table→ncols
      WRITE fp table→items[i][j]
      j += 1
    ENDWHILE
    WRITE fp "\n"
    i += 1
  ENDWHILE
  CLOSE fp
END

```



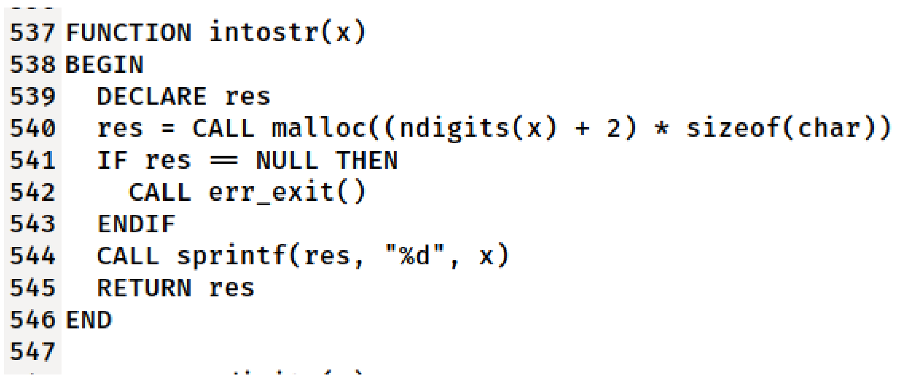
30. strtoint function



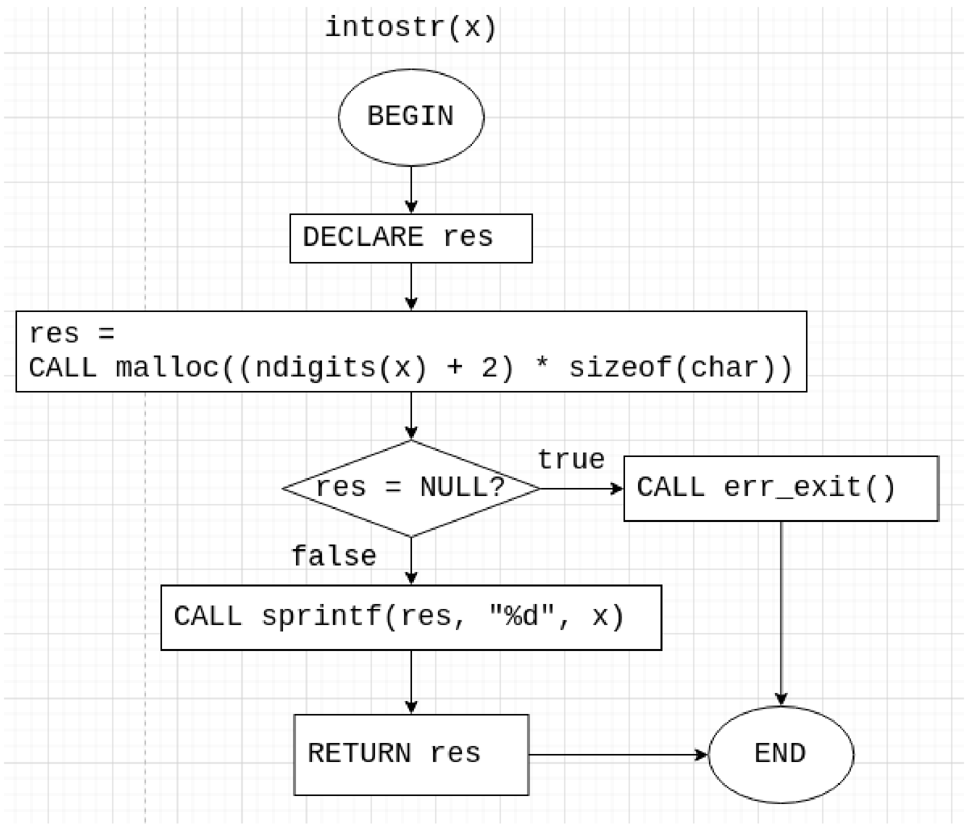
```
FUNCTION strtoint(str)
BEGIN
  DECLARE result
  CALL sscanf(str, "%d", &result)
  RETURN result
END

```



31. intostr function

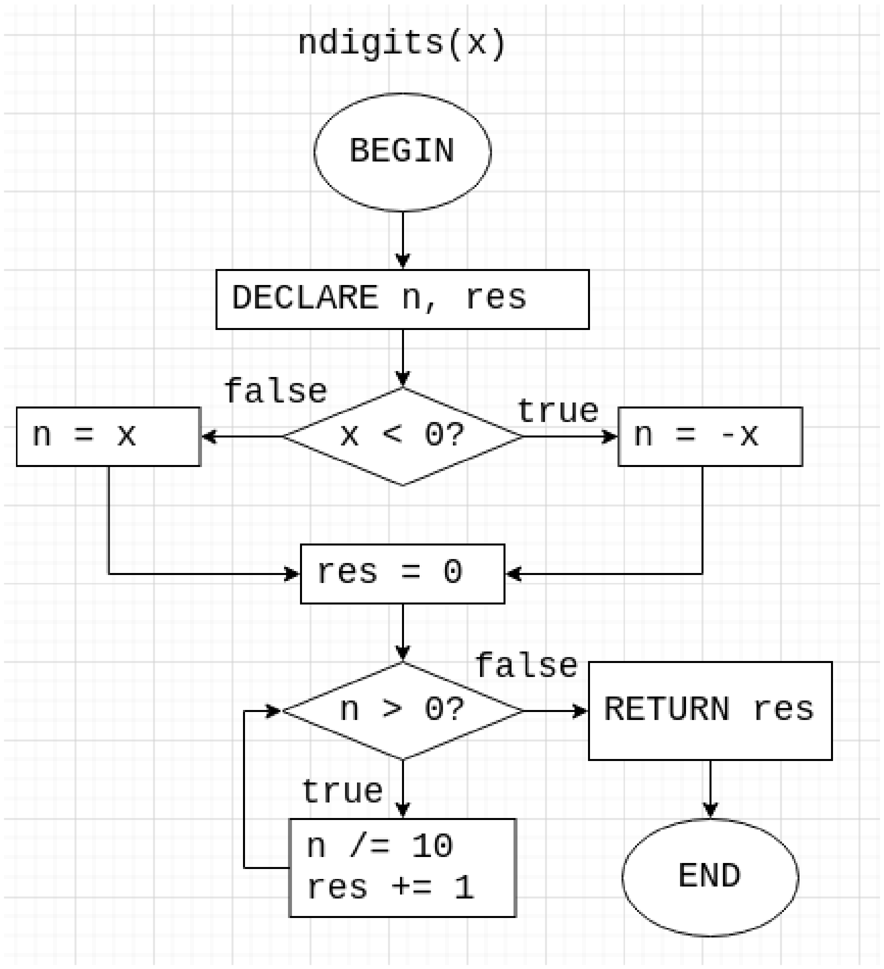


```
FUNCTION intostr(x)
BEGIN
  DECLARE res
  res = CALL malloc((ndigits(x) + 2) * sizeof(char))
  IF res == NULL THEN
    CALL err_exit()
  ENDIF
  CALL sprintf(res, "%d", x)
  RETURN res
END
```



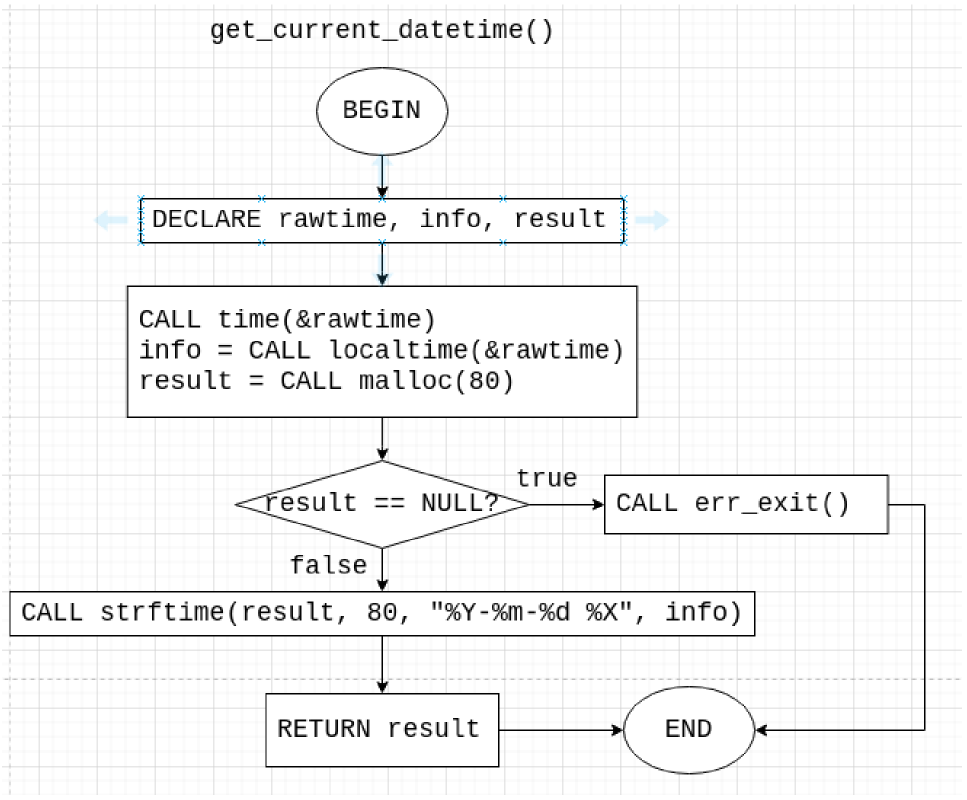
32. ndigits function

```
FUNCTION ndigits(x)
BEGIN
  DECLARE n, res
  IF x < 0 THEN
    n  = -x
  ELSE
    n = x
  ENDIF
  res = 0
  WHILE n > 0
    n /= 10
    res += 1
  ENWDHILE
  RETURN res
END

```



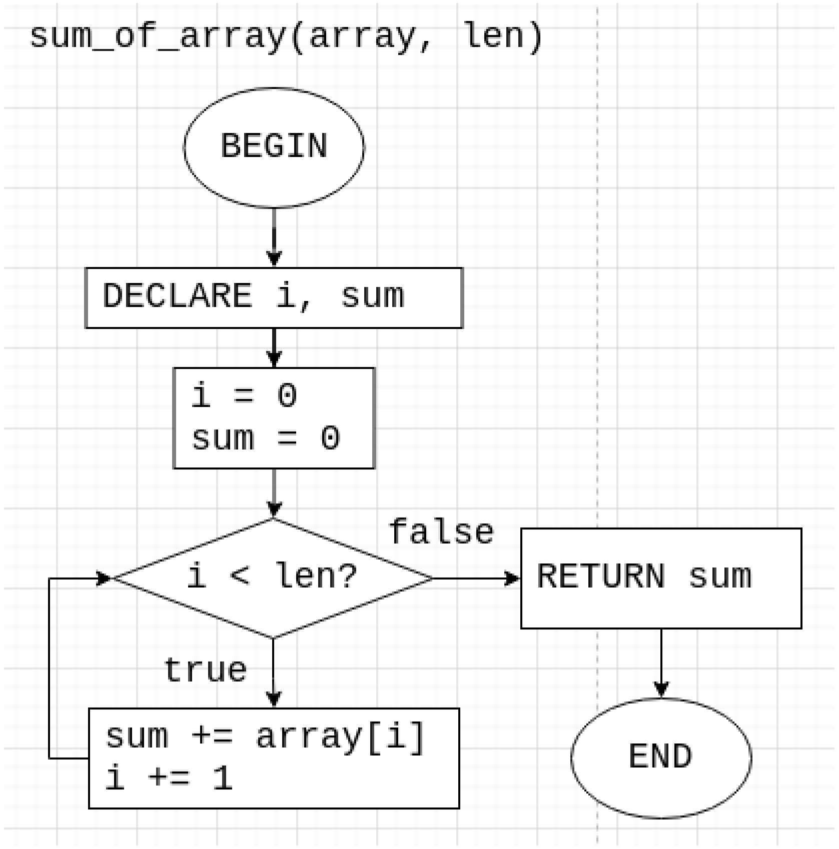
33. get_current_datetime function

```
FUNCTION get_current_datetime()
BEGIN
  DECLARE rawtime, info, result
  CALL time(&rawtime)
  info = CALL localtime(&rawtime)
  result = CALL malloc(80)
  IF result == NULL THEN
    CALL err_exit()
  ENDIF
  CALL strftime(result, 80, "%Y-%m-%d %X", info)
  RETURN result
END
```



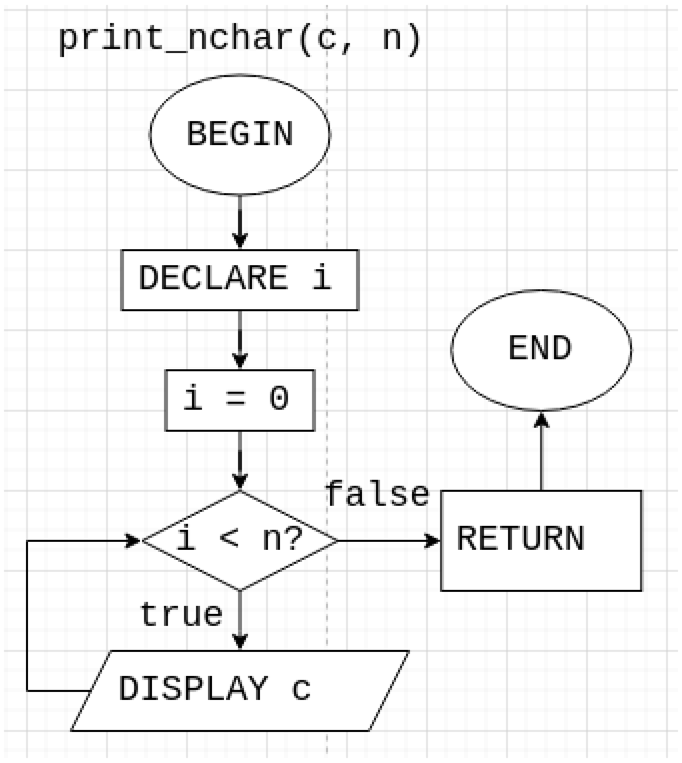
34. sum_of_array function

```
FUNCTION sum_of_array(array, len)
BEGIN
  DECLARE i, sum
  i = 0
  sum = 0
  WHILE i < len
    sum += array[i]
    i += 1
  ENDWHILE
  RETURN sum
END

```

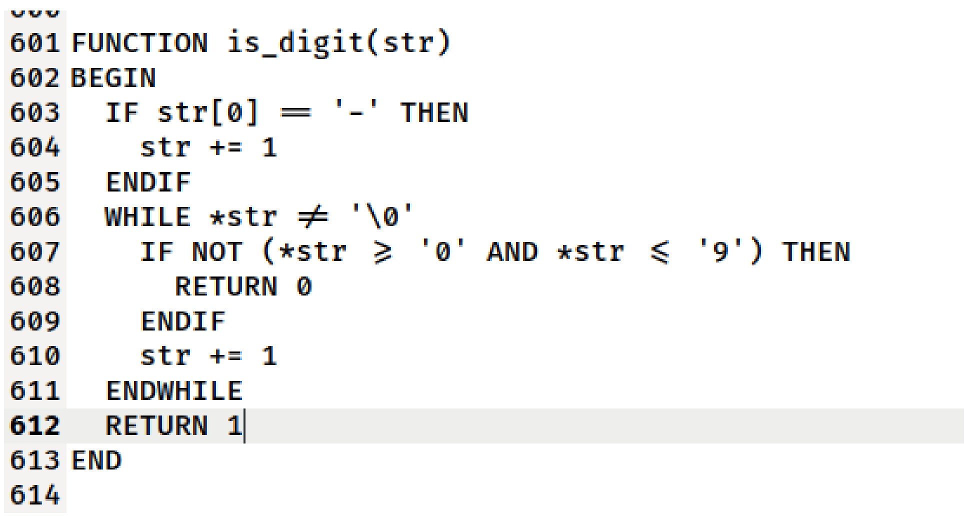


35. print_nchar function

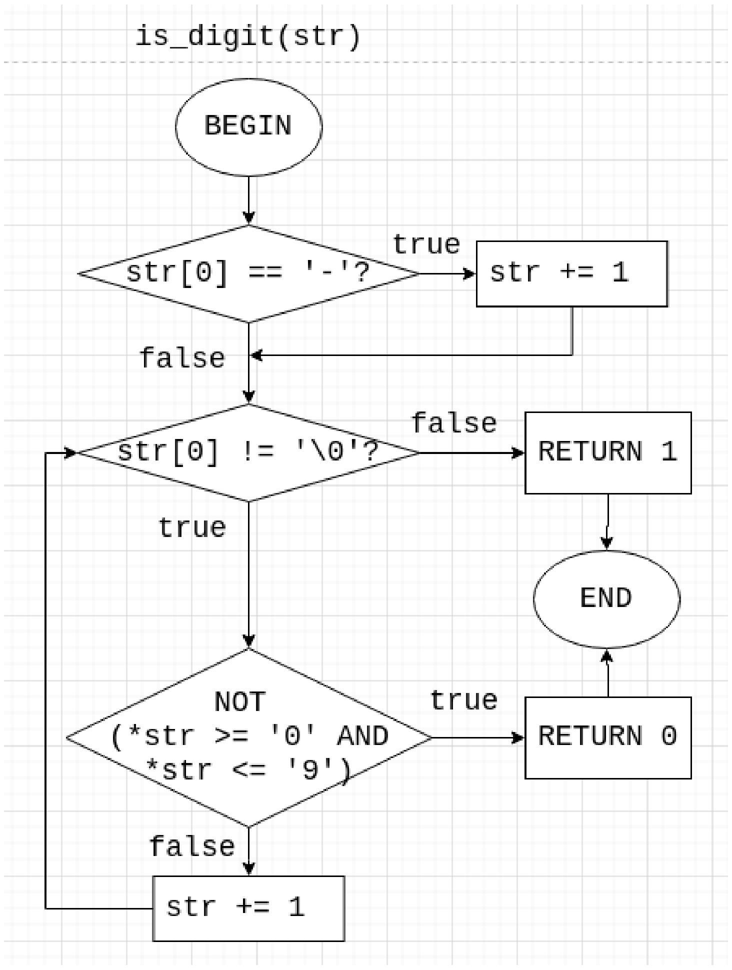
```
FUNCTION print_nchar(c, n)
BEGIN
  DECLARE i
  i = 0
  WHILE i < n
    DISPLAY c
  ENDWHILE
END

```



36. is_digit function

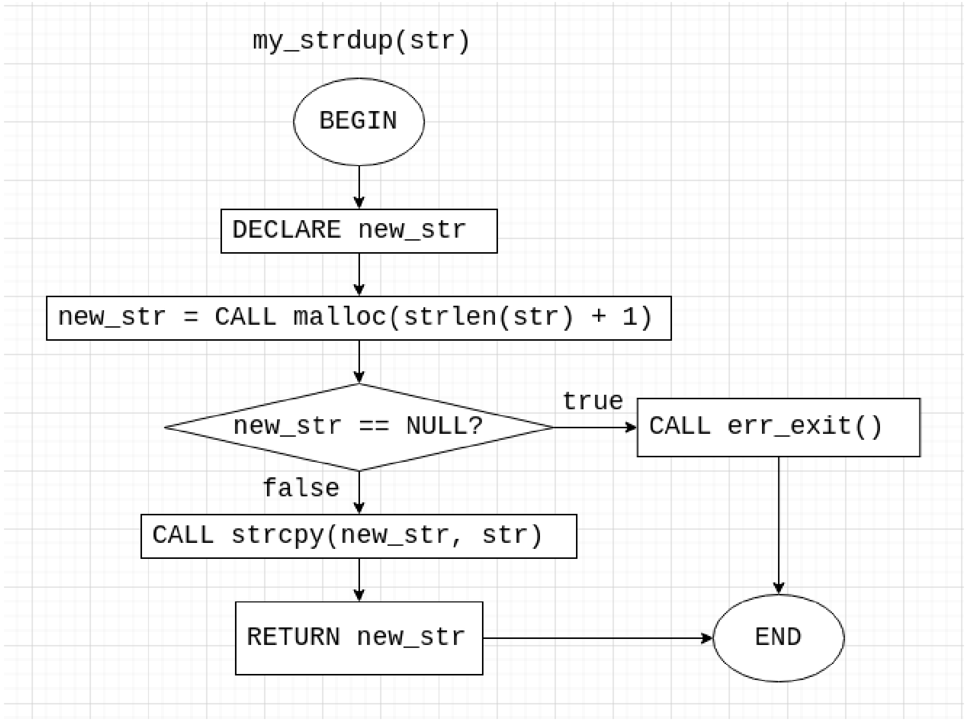


```
FUNCTION is_digit(str)
BEGIN
  IF str[0] == '-' THEN
    str += 1
  ENDIF
  WHILE *str ≠ '\0'
    IF NOT (*str ≥ '0' AND *str ≤ '9') THEN
      RETURN 0
    ENDIF
    str += 1
  ENDWHILE
  RETURN 1
END
```



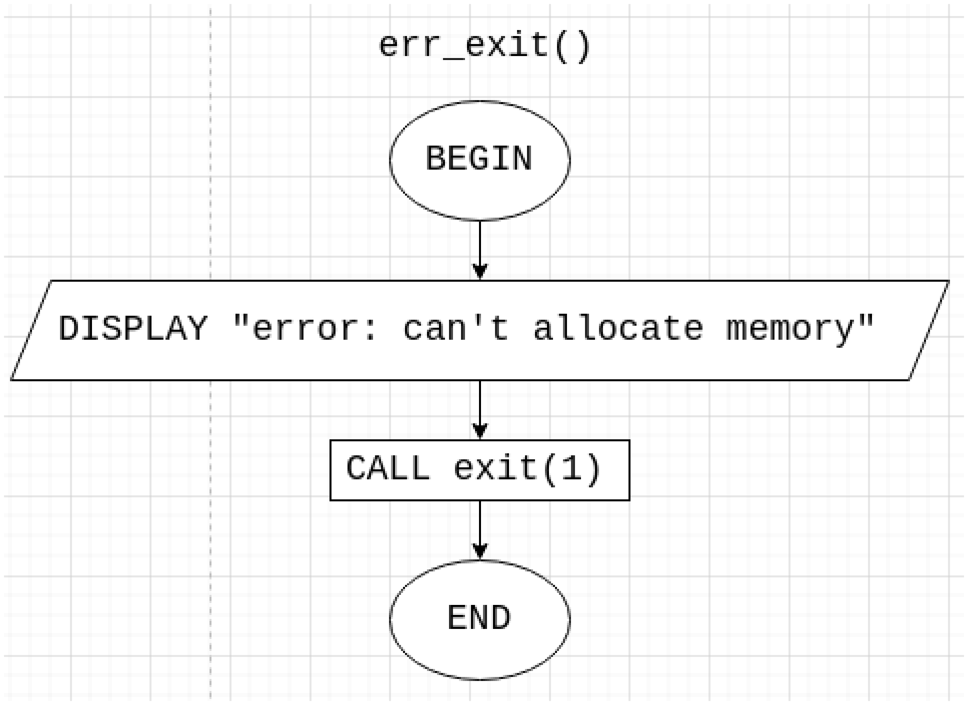
37. my_strdup function

```
FUNCTION my_strdup(str)
BEGIN
  DECLARE new_str
  new_str = CALL malloc(strlen(str) + 1)
  IF new_str == NULL THEN
    CALL err_exit()
  ENDIF
  CALL strcpy(new_str, str)
  RETURN new_str
END
```



38. err_exit function

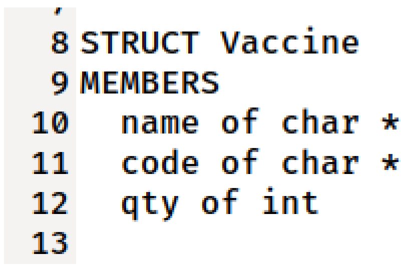
```
FUNCTION err_exit()
BEGIN
  DISPLAY "error: can't allocate memory"
  CALL exit(1)
END

```



39. Table struct

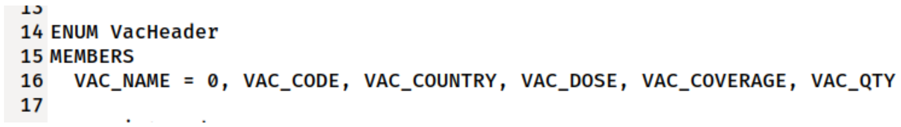
```
STRUCT Table
MEMBERS
  filename of char *
  nrows of int
  ncols of int
  items of char **

```

40. Vaccine struct



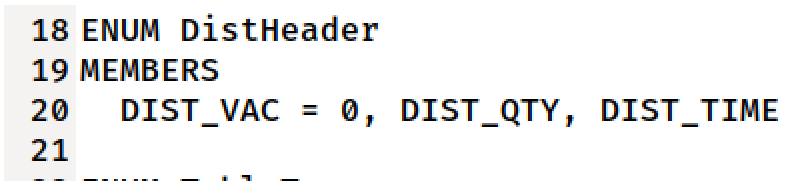
```
STRUCT Vaccine
MEMBERS
  name of char *
  code of char *
  qty of int

```

41. VacHeader enum



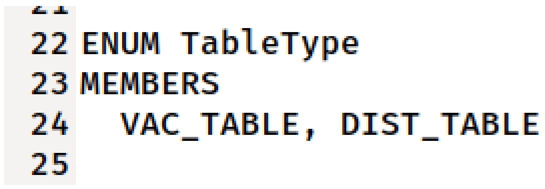
```
ENUM VacHeader
MEMBERS
  VAC_NAME = 0, VAC_CODE, VAC_COUNTRY, VAC_DOSE, VAC_COVERAGE, VAC_QTY

```

42. DistHeader enum



```
ENUM DistHeader
MEMBERS
  DIST_VAC = 0, DIST_QTY, DIST_TIME

```

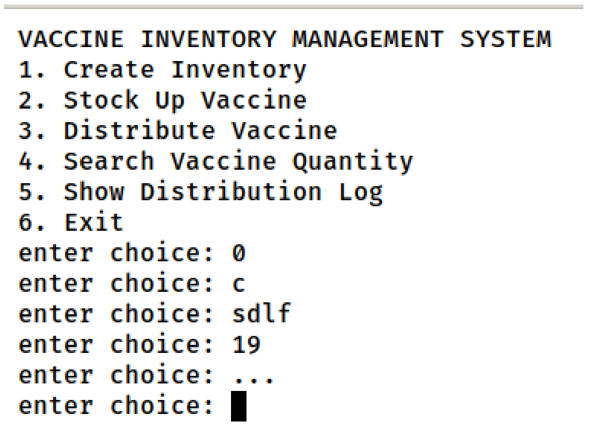
43. TableType enum



```
ENUM TableType
MEMBERS
  VAC_TABLE, DIST_TABLE

```

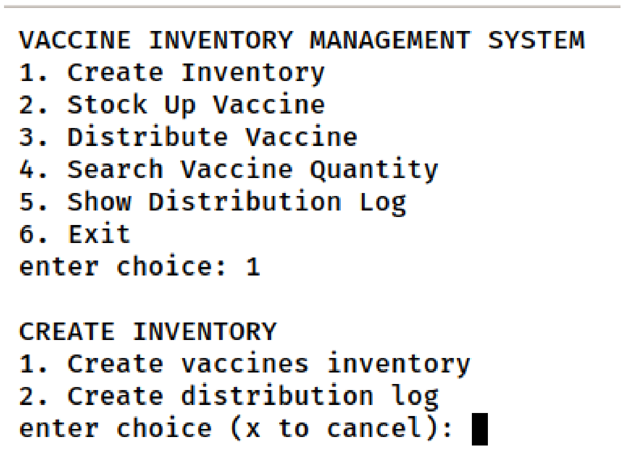
Sample Input and Outputs

Below are the sample inputs and outputs for the program. The tests below should cover all of the possible test cases for the program.



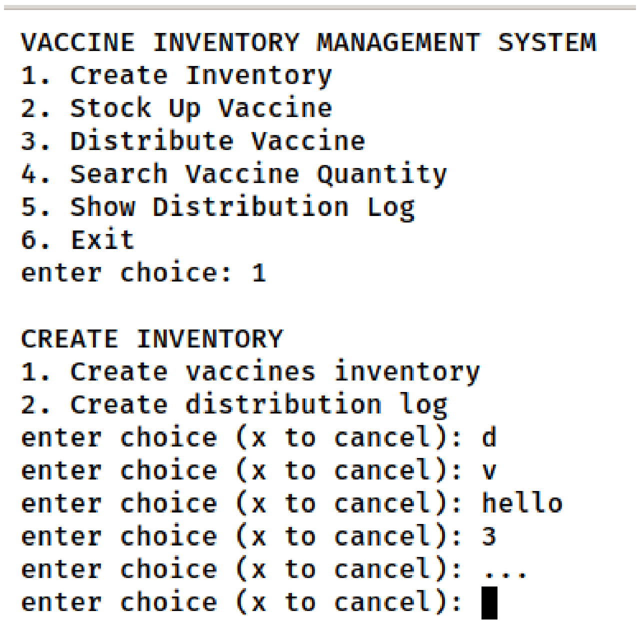
```
VACCINE INVENTORY MANAGEMENT SYSTEM
1. Create Inventory
2. Stock Up Vaccine
3. Distribute Vaccine
4. Search Vaccine Quantity
5. Show Distribution Log
6. Exit
enter choice: 0
enter choice: c
enter choice: sdlf
enter choice: 19
enter choice: ...
enter choice: █
```

This is the main menu of the program. The only valid inputs are the numbers 1 to 6. Any other inputs will result in the input prompt being repeated until a valid input is given.



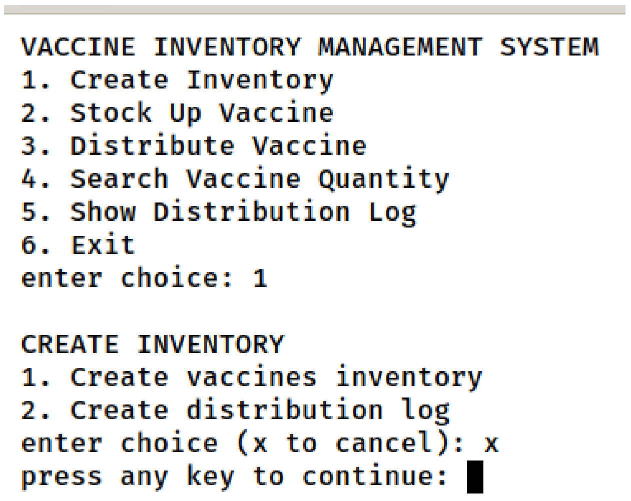
```
VACCINE INVENTORY MANAGEMENT SYSTEM
1. Create Inventory
2. Stock Up Vaccine
3. Distribute Vaccine
4. Search Vaccine Quantity
5. Show Distribution Log
6. Exit
enter choice: 1

CREATE INVENTORY
1. Create vaccines inventory
2. Create distribution log
enter choice (x to cancel): █
```

This is the inventory creation functionality of the system. Users can get to this after entering '1' in the main menu input prompt. This menu allows users to either create the vaccines inventory or the distribution log.



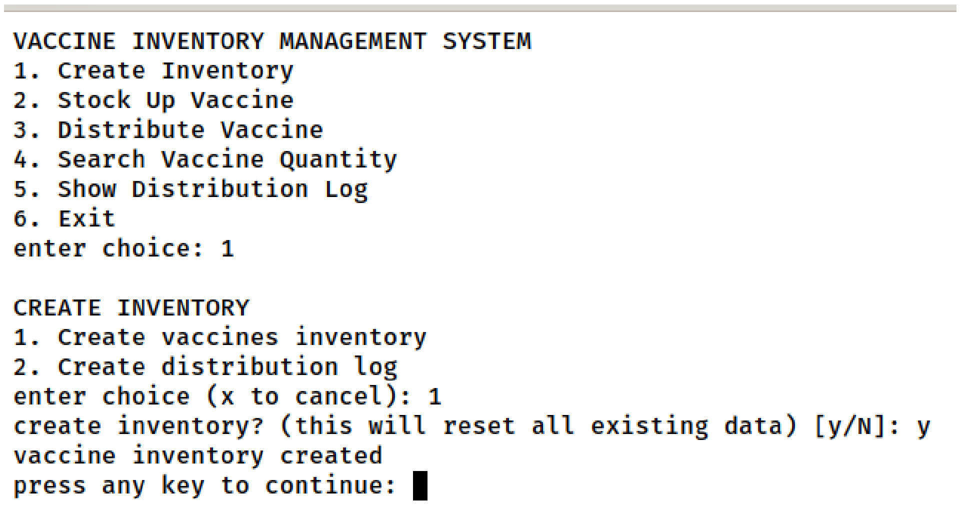
```
VACCINE INVENTORY MANAGEMENT SYSTEM
1. Create Inventory
2. Stock Up Vaccine
3. Distribute Vaccine
4. Search Vaccine Quantity
5. Show Distribution Log
6. Exit
enter choice: 1

CREATE INVENTORY
1. Create vaccines inventory
2. Create distribution log
enter choice (x to cancel): d
enter choice (x to cancel): v
enter choice (x to cancel): hello
enter choice (x to cancel): 3
enter choice (x to cancel): ...
enter choice (x to cancel): 
```

Any invalid inputs given to the prompt will result in it being repeated until a valid input is given.



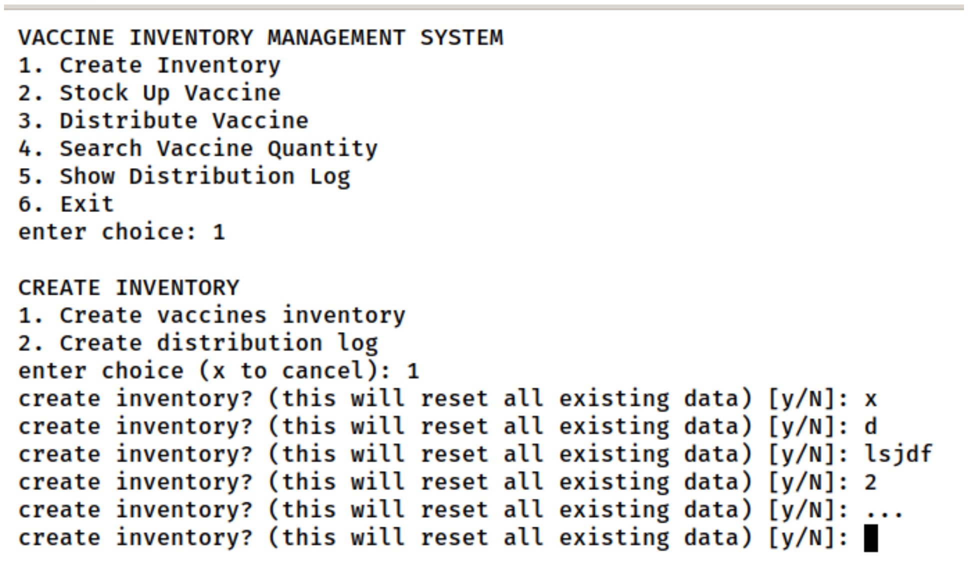
```
VACCINE INVENTORY MANAGEMENT SYSTEM
1. Create Inventory
2. Stock Up Vaccine
3. Distribute Vaccine
4. Search Vaccine Quantity
5. Show Distribution Log
6. Exit
enter choice: 1

CREATE INVENTORY
1. Create vaccines inventory
2. Create distribution log
enter choice (x to cancel): x
press any key to continue: 
```

If given ‘x’ as input, the program will return to the main menu. Before returning, the program will prompt the user to press any key to continue, because returning to the main menu will clear all outputs shown when using a menu item.



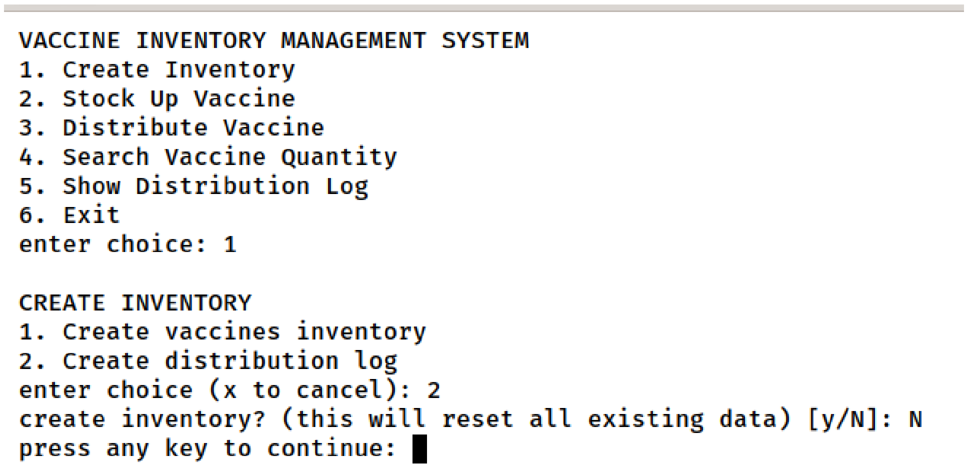
```
VACCINE INVENTORY MANAGEMENT SYSTEM
1. Create Inventory
2. Stock Up Vaccine
3. Distribute Vaccine
4. Search Vaccine Quantity
5. Show Distribution Log
6. Exit
enter choice: 1

CREATE INVENTORY
1. Create vaccines inventory
2. Create distribution log
enter choice (x to cancel): 1
create inventory? (this will reset all existing data) [y/N]: y
vaccine inventory created
press any key to continue:
```

If given ‘1’ as the input, the program will print a confirmation prompt to the user before committing the change. If the user confirms, the vaccines.txt file will be created and the message “vaccine inventory created” will be printed. Again, before returning to the main menu, the wait for input prompt is shown.



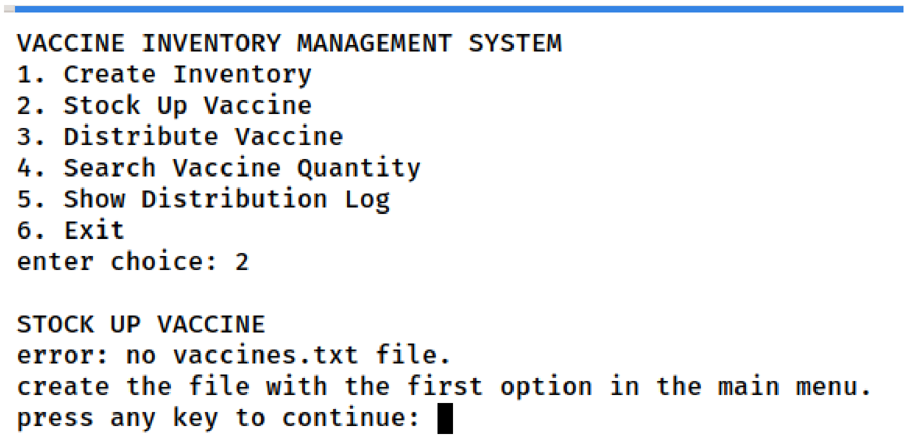
```
VACCINE INVENTORY MANAGEMENT SYSTEM
1. Create Inventory
2. Stock Up Vaccine
3. Distribute Vaccine
4. Search Vaccine Quantity
5. Show Distribution Log
6. Exit
enter choice: 1

CREATE INVENTORY
1. Create vaccines inventory
2. Create distribution log
enter choice (x to cancel): 1
create inventory? (this will reset all existing data) [y/N]: x
create inventory? (this will reset all existing data) [y/N]: d
create inventory? (this will reset all existing data) [y/N]: lsjdf
create inventory? (this will reset all existing data) [y/N]: 2
create inventory? (this will reset all existing data) [y/N]: ...
create inventory? (this will reset all existing data) [y/N]:
```

The only valid inputs for the confirmation prompt are ‘y’, ‘Y’, ‘n’, and ‘N’. Any other inputs will result in the program asking for input until a valid one is given.



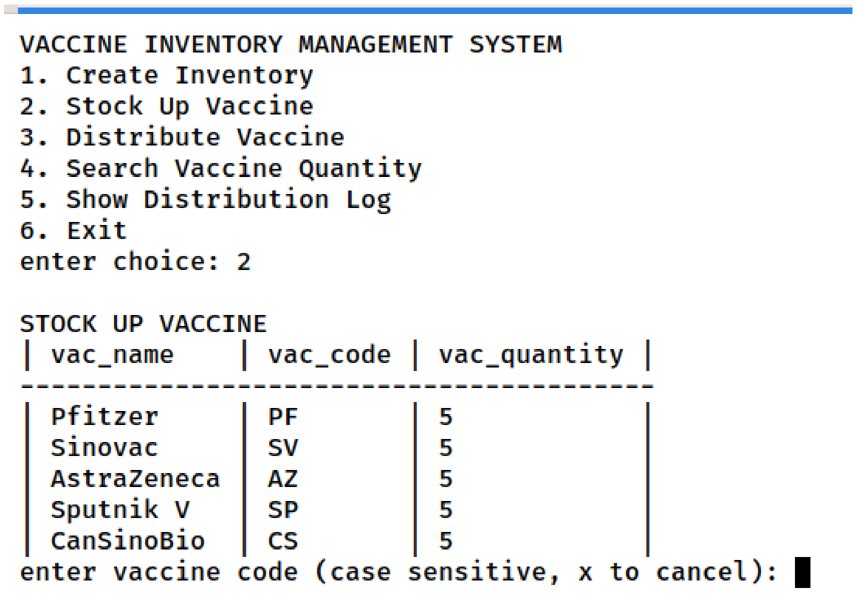
```
VACCINE INVENTORY MANAGEMENT SYSTEM
1. Create Inventory
2. Stock Up Vaccine
3. Distribute Vaccine
4. Search Vaccine Quantity
5. Show Distribution Log
6. Exit
enter choice: 1

CREATE INVENTORY
1. Create vaccines inventory
2. Create distribution log
enter choice (x to cancel): 2
create inventory? (this will reset all existing data) [y/N]: N
press any key to continue:
```

The input/output handling of the “Create vaccines inventory” and the “Create distribution log” are the same. If ‘n’ or ‘N’ is given as input, the program will return to the main menu without writing any files.



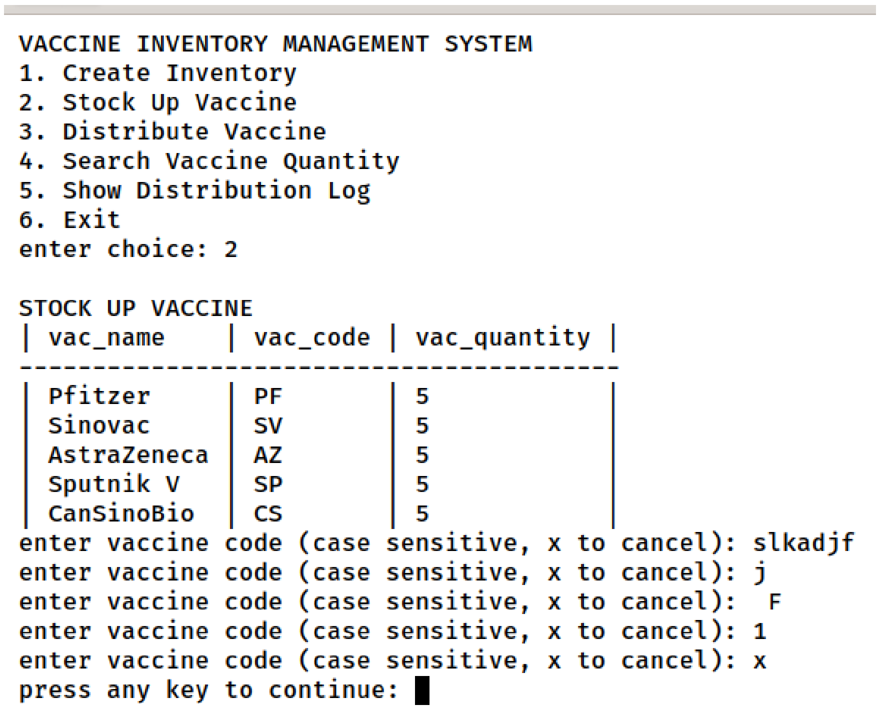
```
VACCINE INVENTORY MANAGEMENT SYSTEM
1. Create Inventory
2. Stock Up Vaccine
3. Distribute Vaccine
4. Search Vaccine Quantity
5. Show Distribution Log
6. Exit
enter choice: 2

STOCK UP VACCINE
error: no vaccines.txt file.
create the file with the first option in the main menu.
press any key to continue:
```

This is the stock up vaccine functionality. If the file vaccines.txt has not been created, the program will print a message and return to the main menu.



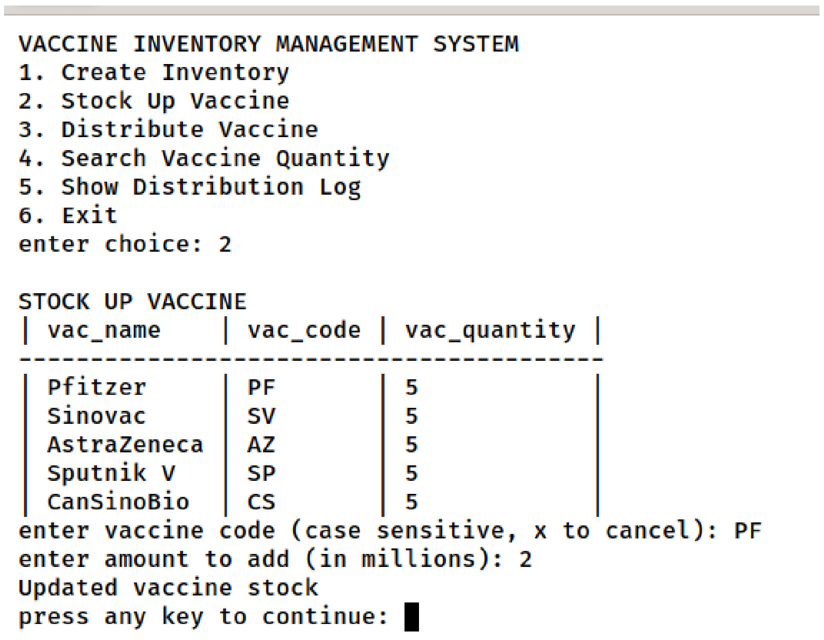
```
VACCINE INVENTORY MANAGEMENT SYSTEM
1. Create Inventory
2. Stock Up Vaccine
3. Distribute Vaccine
4. Search Vaccine Quantity
5. Show Distribution Log
6. Exit
enter choice: 2

STOCK UP VACCINE
| vac_name     | vac_code | vac_quantity |
-----------------------------------------
| Pfitzer      | PF       | 5            |
  Sinovac      | SV       | 5
  AstraZeneca  | AZ       | 5
  Sputnik V    | SP       | 5
| CanSinoBio   | CS       | 5            |
enter vaccine code (case sensitive, x to cancel): 
```

This is the output for stock up vaccine if the vaccines.txt file has been created. The program will print the file in a tabular format, listing the names, codes, and quantities (in millions) of each vaccine. The only valid inputs for this prompt are the vaccine codes shown on the screen and ‘x’.



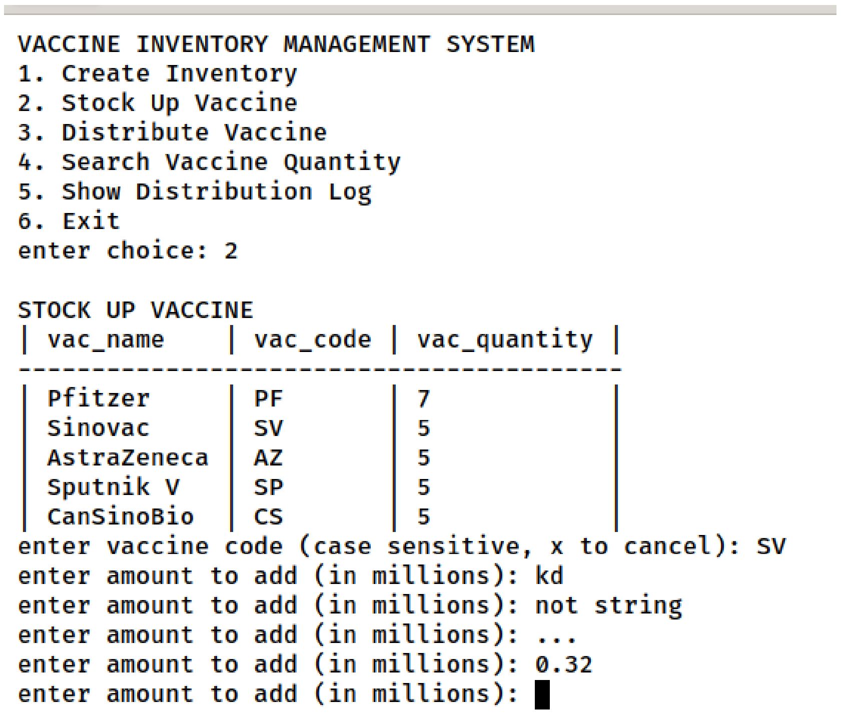
```
VACCINE INVENTORY MANAGEMENT SYSTEM
1. Create Inventory
2. Stock Up Vaccine
3. Distribute Vaccine
4. Search Vaccine Quantity
5. Show Distribution Log
6. Exit
enter choice: 2

STOCK UP VACCINE
| vac_name     | vac_code | vac_quantity |
-----------------------------------------
| Pfitzer      | PF       | 5            |
  Sinovac      | SV       | 5
  AstraZeneca  | AZ       | 5
  Sputnik V    | SP       | 5
| CanSinoBio   | CS       | 5            |
enter vaccine code (case sensitive, x to cancel): slkadjf
enter vaccine code (case sensitive, x to cancel): j
enter vaccine code (case sensitive, x to cancel):  F
enter vaccine code (case sensitive, x to cancel): 1
enter vaccine code (case sensitive, x to cancel): x
press any key to continue: 
```

Above is the output for invalid inputs and the valid ‘x’ input.



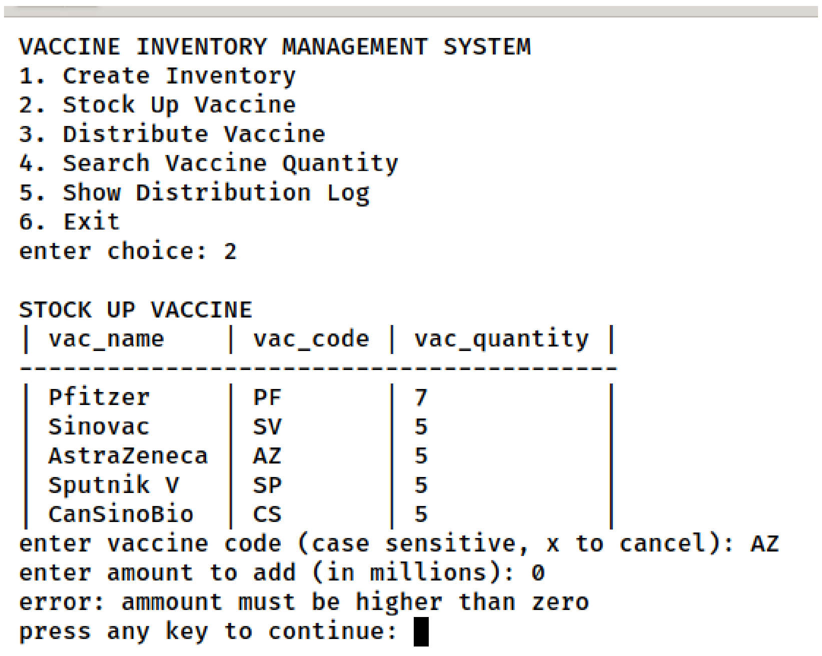
```
VACCINE INVENTORY MANAGEMENT SYSTEM
1. Create Inventory
2. Stock Up Vaccine
3. Distribute Vaccine
4. Search Vaccine Quantity
5. Show Distribution Log
6. Exit
enter choice: 2

STOCK UP VACCINE
| vac_name    | vac_code | vac_quantity |
-----------------------------------------
| Pfitzer     | PF       | 5            |
  Sinovac     | SV       | 5
  AstraZeneca | AZ       | 5
  Sputnik V   | SP       | 5
| CanSinoBio  | CS       | 5            |
enter vaccine code (case sensitive, x to cancel): PF
enter amount to add (in millions): 2
Updated vaccine stock
press any key to continue: 
```

If a valid vaccine code is given, the program will prompt for the number of vaccines to add (in millions). If a valid integer is given, the file will be updated and the message "updated vaccine stock" will be printed. The user will then be redirected back to the main menu.



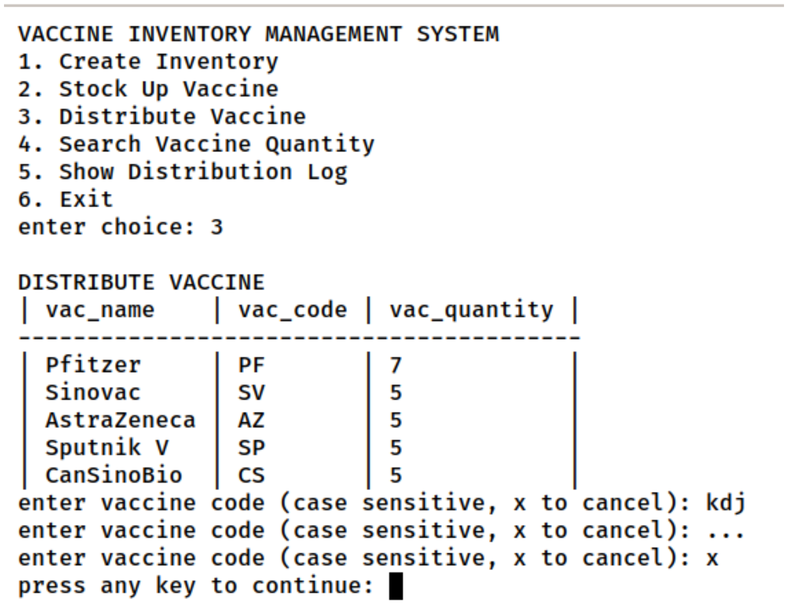
```
VACCINE INVENTORY MANAGEMENT SYSTEM
1. Create Inventory
2. Stock Up Vaccine
3. Distribute Vaccine
4. Search Vaccine Quantity
5. Show Distribution Log
6. Exit
enter choice: 2

STOCK UP VACCINE
| vac_name    | vac_code | vac_quantity |
-----------------------------------------
| Pfitzer     | PF       | 7            |
  Sinovac     | SV       | 5
  AstraZeneca | AZ       | 5
  Sputnik V   | SP       | 5
| CanSinoBio  | CS       | 5            |
enter vaccine code (case sensitive, x to cancel): SV
enter amount to add (in millions): kd
enter amount to add (in millions): not string
enter amount to add (in millions): ...
enter amount to add (in millions): 0.32
enter amount to add (in millions): 
```

As shown in the screenshot above, the quantity for the Pfitzer vaccine has been updated. If a string, floating point number, or any other non-integer value is given the program will keep asking for input until a valid value is given.



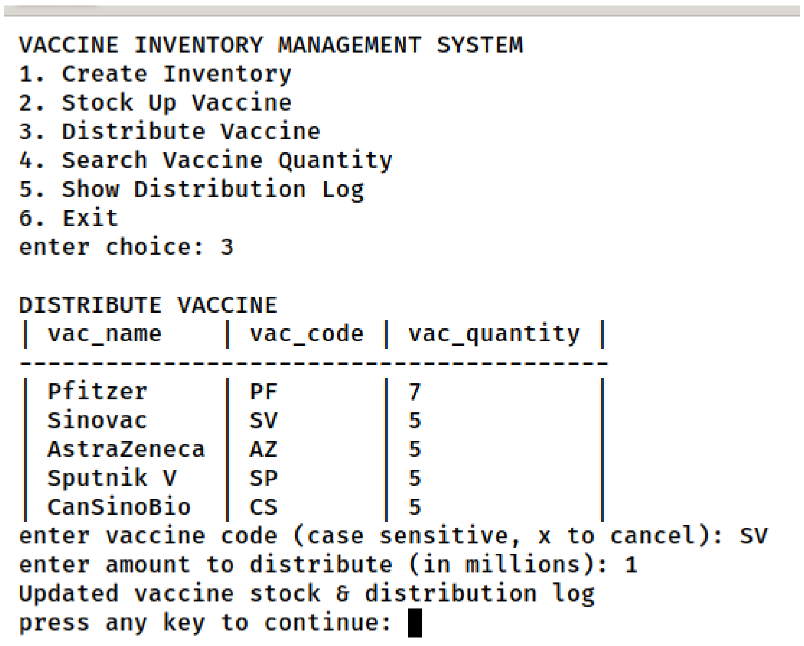
```
VACCINE INVENTORY MANAGEMENT SYSTEM
1. Create Inventory
2. Stock Up Vaccine
3. Distribute Vaccine
4. Search Vaccine Quantity
5. Show Distribution Log
6. Exit
enter choice: 2

STOCK UP VACCINE
| vac_name     | vac_code | vac_quantity |
-----------------------------------------
| Pfitzer      | PF       | 7            |
| Sinovac      | SV       | 5            |
| AstraZeneca  | AZ       | 5            |
| Sputnik V    | SP       | 5            |
| CanSinoBio   | CS       | 5            |
enter vaccine code (case sensitive, x to cancel): AZ
enter amount to add (in millions): 0
error: ammount must be higher than zero
press any key to continue: █
```

If a zero or a negative integer is given, the program will print a message to the user and return to the main menu.



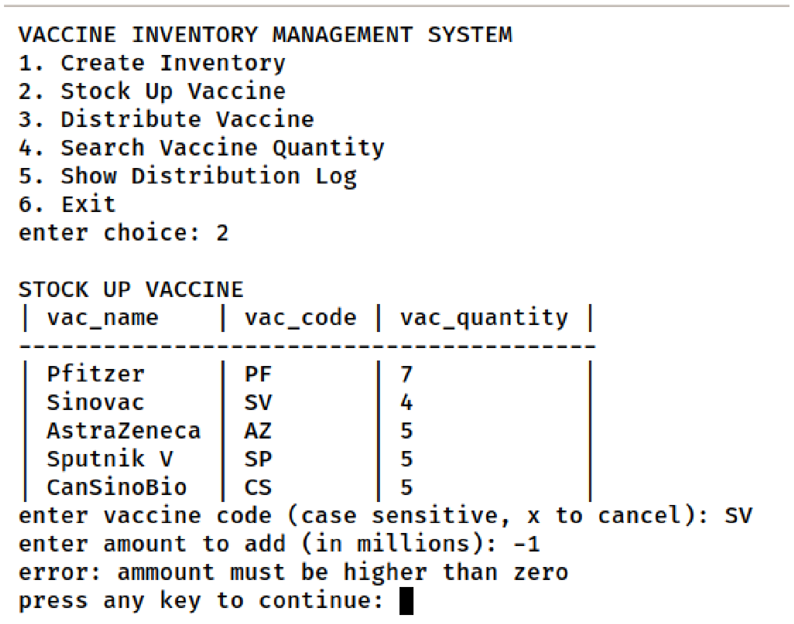
```
VACCINE INVENTORY MANAGEMENT SYSTEM
1. Create Inventory
2. Stock Up Vaccine
3. Distribute Vaccine
4. Search Vaccine Quantity
5. Show Distribution Log
6. Exit
enter choice: 3

DISTRIBUTE VACCINE
| vac_name     | vac_code | vac_quantity |
-----------------------------------------
| Pfitzer      | PF       | 7            |
  Sinovac      | SV       | 5            |
  AstraZeneca  | AZ       | 5            |
  Sputnik V    | SP       | 5            |
| CanSinoBio   | CS       | 5            |
enter vaccine code (case sensitive, x to cancel): kdj
enter vaccine code (case sensitive, x to cancel): ...
enter vaccine code (case sensitive, x to cancel): x
press any key to continue: █
```

This is the menu for the distribute vaccine functionality. Just like the option for adding vaccines, the details of the vaccine are printed in a tabular format, and the only valid inputs are the printed vaccine codes and 'x'.



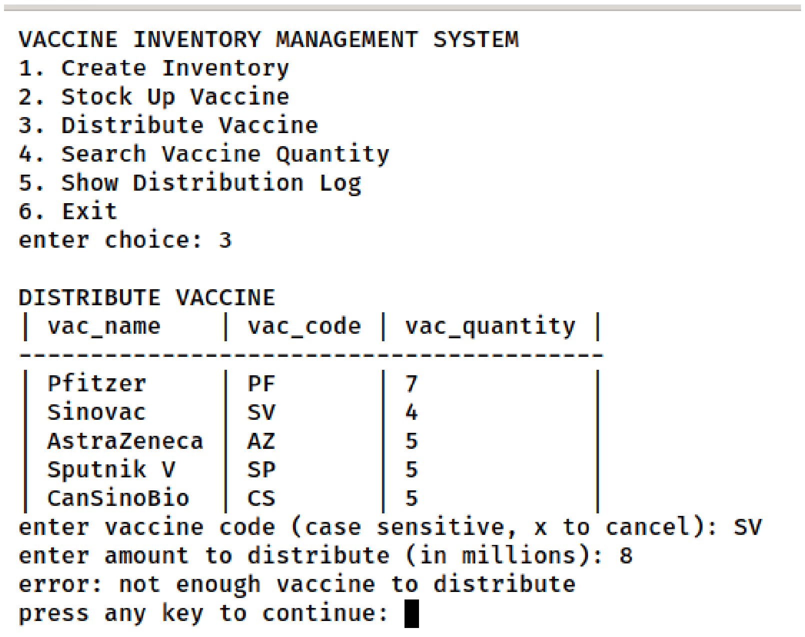
```
VACCINE INVENTORY MANAGEMENT SYSTEM
1. Create Inventory
2. Stock Up Vaccine
3. Distribute Vaccine
4. Search Vaccine Quantity
5. Show Distribution Log
6. Exit
enter choice: 3

DISTRIBUTE VACCINE
| vac_name     | vac_code | vac_quantity |
-----------------------------------------
| Pfitzer      | PF       | 7            |
| Sinovac      | SV       | 5            |
| AstraZeneca  | AZ       | 5            |
| Sputnik V    | SP       | 5            |
| CanSinoBio   | CS       | 5            |
enter vaccine code (case sensitive, x to cancel): SV
enter amount to distribute (in millions): 1
Updated vaccine stock & distribution log
press any key to continue: █
```

If a valid vaccine code is given, the program will prompt for the number of vaccines to distribute (in millions). If a valid integer is given, the program will update the vaccines.txt file as well as the dist.txt file. It will then print the message "Updated vaccine stock & distribution log", and return to main menu.



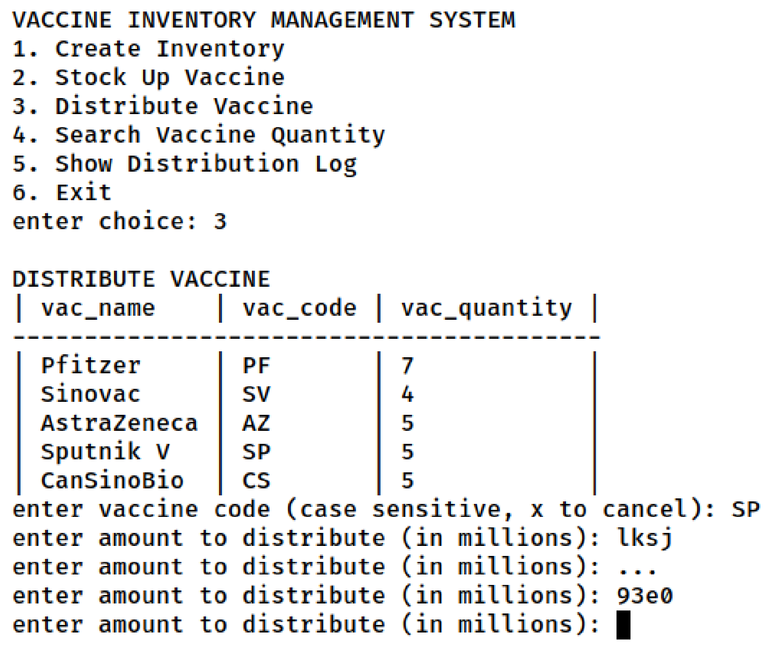
```
VACCINE INVENTORY MANAGEMENT SYSTEM
1. Create Inventory
2. Stock Up Vaccine
3. Distribute Vaccine
4. Search Vaccine Quantity
5. Show Distribution Log
6. Exit
enter choice: 2

STOCK UP VACCINE
| vac_name      | vac_code | vac_quantity |
-----------------------------------------
| Pfitzer       | PF       | 7            |
| Sinovac       | SV       | 4            |
| AstraZeneca   | AZ       | 5            |
| Sputnik V     | SP       | 5            |
| CanSinoBio    | CS       | 5            |
enter vaccine code (case sensitive, x to cancel): SV
enter amount to add (in millions): -1
error: ammount must be higher than zero
press any key to continue: 
```

As you can see, the quantity for the Sinovac vaccine has been updated. If a valid integer is given but it is zero or a negative number, the program will print a message and return to the main menu.



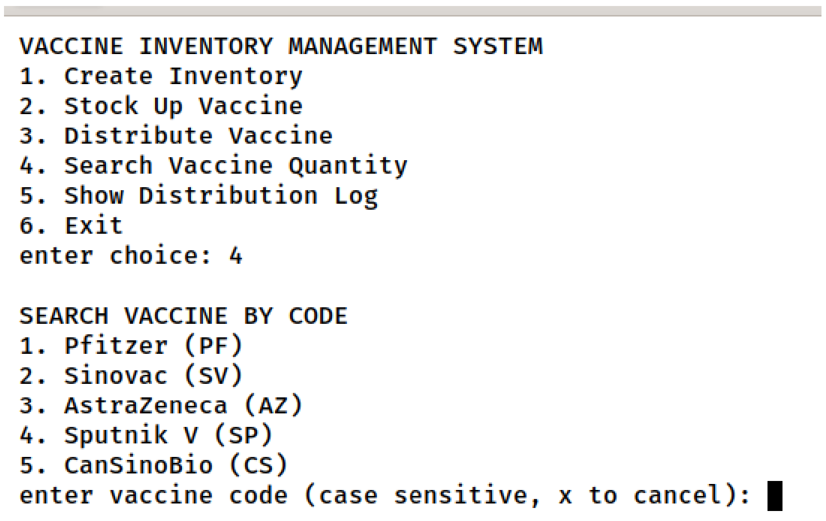
```
VACCINE INVENTORY MANAGEMENT SYSTEM
1. Create Inventory
2. Stock Up Vaccine
3. Distribute Vaccine
4. Search Vaccine Quantity
5. Show Distribution Log
6. Exit
enter choice: 3

DISTRIBUTE VACCINE
| vac_name      | vac_code | vac_quantity |
-----------------------------------------
| Pfitzer       | PF       | 7            |
| Sinovac       | SV       | 4            |
| AstraZeneca   | AZ       | 5            |
| Sputnik V     | SP       | 5            |
| CanSinoBio    | CS       | 5            |
enter vaccine code (case sensitive, x to cancel): SV
enter amount to distribute (in millions): 8
error: not enough vaccine to distribute
press any key to continue: 
```

If the number of vaccines to distribute is greater than the available quantity, the program will display a message and return to the main menu.



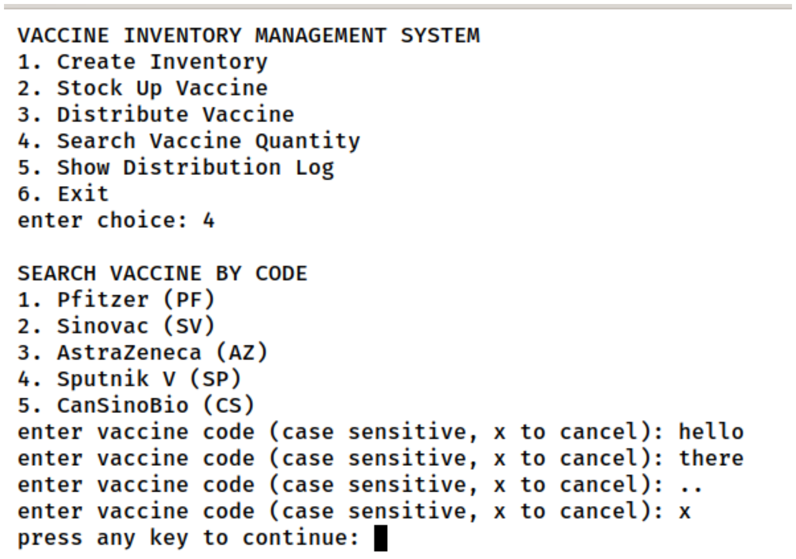
```
VACCINE INVENTORY MANAGEMENT SYSTEM
1. Create Inventory
2. Stock Up Vaccine
3. Distribute Vaccine
4. Search Vaccine Quantity
5. Show Distribution Log
6. Exit
enter choice: 3

DISTRIBUTE VACCINE
| vac_name    | vac_code | vac_quantity |
-----------------------------------------
| Pfitzer     | PF       | 7            |
| Sinovac     | SV       | 4            |
| AstraZeneca | AZ       | 5            |
| Sputnik V   | SP       | 5            |
| CanSinoBio  | CS       | 5            |
enter vaccine code (case sensitive, x to cancel): SP
enter amount to distribute (in millions): lksj
enter amount to distribute (in millions): ...
enter amount to distribute (in millions): 93e0
enter amount to distribute (in millions): █
```

Just like “Stock up vaccine”, any non-valid input will result in the program asking for input until a valid response is given.



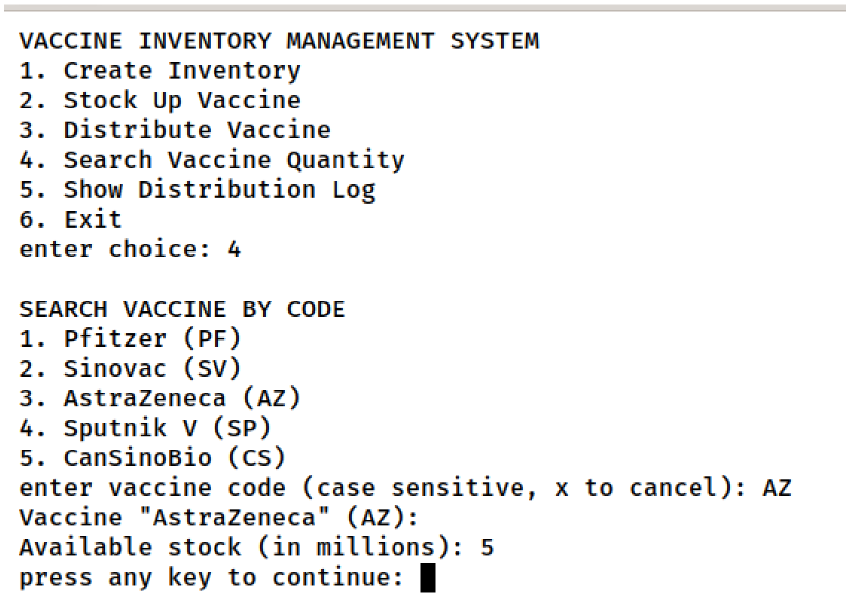
```
VACCINE INVENTORY MANAGEMENT SYSTEM
1. Create Inventory
2. Stock Up Vaccine
3. Distribute Vaccine
4. Search Vaccine Quantity
5. Show Distribution Log
6. Exit
enter choice: 4

SEARCH VACCINE BY CODE
1. Pfitzer (PF)
2. Sinovac (SV)
3. AstraZeneca (AZ)
4. Sputnik V (SP)
5. CanSinoBio (CS)
enter vaccine code (case sensitive, x to cancel): █
```

This is the output of the search vaccine functionality. The program first prints the list of vaccines and their codes and asks for input from the user. Just like the previous 2 option, the only valid inputs are the displayed vaccine codes and ‘x’.



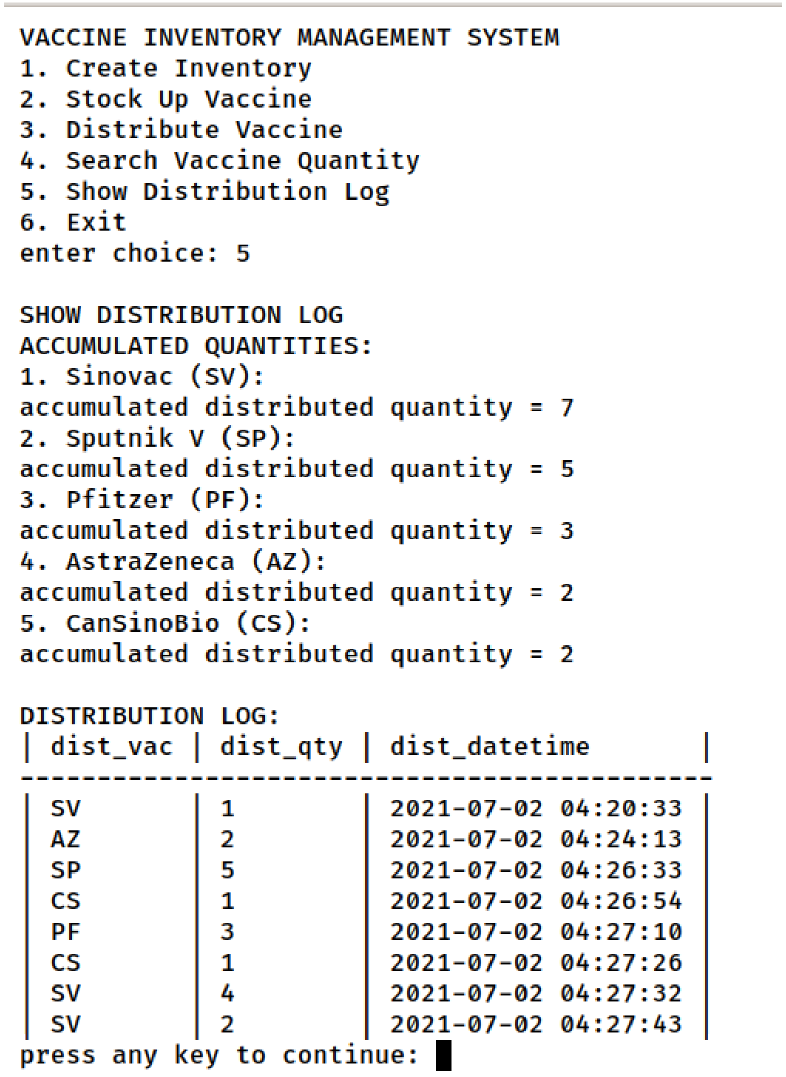
```
VACCINE INVENTORY MANAGEMENT SYSTEM
1. Create Inventory
2. Stock Up Vaccine
3. Distribute Vaccine
4. Search Vaccine Quantity
5. Show Distribution Log
6. Exit
enter choice: 4

SEARCH VACCINE BY CODE
1. Pfitzer (PF)
2. Sinovac (SV)
3. AstraZeneca (AZ)
4. Sputnik V (SP)
5. CanSinoBio (CS)
enter vaccine code (case sensitive, x to cancel): hello
enter vaccine code (case sensitive, x to cancel): there
enter vaccine code (case sensitive, x to cancel): ..
enter vaccine code (case sensitive, x to cancel): x
press any key to continue:
```

Just like before, invalid inputs will lead to repetition, and x will return to main menu.



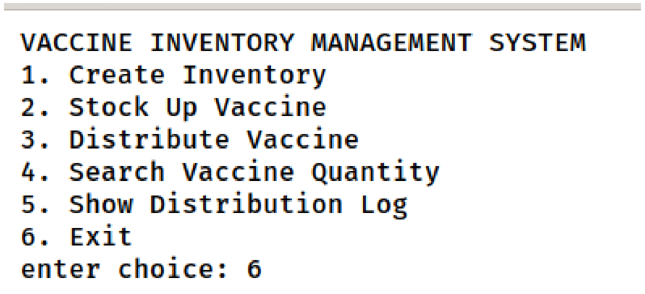
```
VACCINE INVENTORY MANAGEMENT SYSTEM
1. Create Inventory
2. Stock Up Vaccine
3. Distribute Vaccine
4. Search Vaccine Quantity
5. Show Distribution Log
6. Exit
enter choice: 4

SEARCH VACCINE BY CODE
1. Pfitzer (PF)
2. Sinovac (SV)
3. AstraZeneca (AZ)
4. Sputnik V (SP)
5. CanSinoBio (CS)
enter vaccine code (case sensitive, x to cancel): AZ
Vaccine "AstraZeneca" (AZ):
Available stock (in millions): 5
press any key to continue:
```

A valid input will result in the quantity of the vaccine being displayed on the screen. After printing, the program will return to the main menu



```
VACCINE INVENTORY MANAGEMENT SYSTEM
1. Create Inventory
2. Stock Up Vaccine
3. Distribute Vaccine
4. Search Vaccine Quantity
5. Show Distribution Log
6. Exit
enter choice: 5

SHOW DISTRIBUTION LOG
ACCUMULATED QUANTITIES:
1. Sinovac (SV):
accumulated distributed quantity = 7
2. Sputnik V (SP):
accumulated distributed quantity = 5
3. Pfitzer (PF):
accumulated distributed quantity = 3
4. AstraZeneca (AZ):
accumulated distributed quantity = 2
5. CanSinoBio (CS):
accumulated distributed quantity = 2

DISTRIBUTION LOG:
| dist_vac | dist_qty | dist_datetime       |
---------------------------------------------
| SV       | 1        | 2021-07-02 04:20:33 |
| AZ       | 2        | 2021-07-02 04:24:13 |
| SP       | 5        | 2021-07-02 04:26:33 |
| CS       | 1        | 2021-07-02 04:26:54 |
| PF       | 3        | 2021-07-02 04:27:10 |
| CS       | 1        | 2021-07-02 04:27:26 |
| SV       | 4        | 2021-07-02 04:27:32 |
| SV       | 2        | 2021-07-02 04:27:43 |
press any key to continue:
```

This is the output for the show distribution log functionality of the system. The program will print the accumulated distributed quantity of each vaccine, sorted in a descending order. After that, the program will display the distribution log in tabular format. The accumulated quantities match the data from the dist.txt file. The program will then return to the main menu



```
VACCINE INVENTORY MANAGEMENT SYSTEM
1. Create Inventory
2. Stock Up Vaccine
3. Distribute Vaccine
4. Search Vaccine Quantity
5. Show Distribution Log
6. Exit
enter choice: 6
```

Inputting ‘6’ in the main menu prompt will exit the program.

Conclusion

To conclude, the above program has all the functionalities needed for a vaccine inventory management system. Users can create the vaccine inventory and the distribution log, add and remove vaccines from the inventory, log the vaccine distributions in a text file, search vaccine available quantities by their code, and show the sorted accumulated distributed quantities of each vaccine.