

МIНIСТЕРСТВО ОСВIТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**Лабораторна робота №8**

з дисципліни “Основи програмування”

тема “**Динамічна пам'ять**”

|  |  |  |
| --- | --- | --- |
| Виконав(ла)  студент(ка) I курсу  групи КП-92  Мовчан Максим  (*прізвище, ім’я, по батькові*)  варіант №9 |  | Перевірив  “\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 20\_\_\_ р.  викладач  Гадиняк Руслан Анатолійович  (*прізвище, ім’я, по батькові*) |

Київ 2019

**Мета роботи**

Використати динамічну пам'ять для реалізації динамічних масивів та списків зв'язаних вузлів.

**Постановка завдання**

#### **Динамічний список на основі динамічного масиву**

Реалізувати динамічний список **цілих чисел** на основі динамічного масиву, місткість (розмір) якого можна змінювати.

Всі функції даного підзавдання повинні починатись з dynarray\_.

Реалізувати набір чистих функцій для роботи з одним екземпляром такого списку (у фігурних дужках параметри функції):

* init - ініціалізувати пустий список.
* deinit - звільнити всю виділену пам'ять списку.
* clear - очистити список чисел (довжина **списку** має стати 0).
* reserve, {new\_capacity} - змінити місткість динамічного масиву. Якщо нова місткість менша за поточну - нічого не відбувається. Якщо більша - перевиділити динамічну пам'ять з змінити місткість.
* push\_back, {value} - додати новий елемент у кінець списку. Якщо для цієї операції недостатньо поточної місткості динамічного масиву - збільшити його місткіть повторно виділивши пам'ять.
* erase, {index} - видалити елемент списку з даної позиції і зсунути всі праві елементи на одну позицію вліво.

#### Динамічний список зв'язаних вузлів (3 бали)

Реалізувати динамічний список **дробових чисел** на основі зв'язаних вузлів.

Всі функції даного підзавдання повинні починатись з slnode\_.

Реалізувати набір чистих функцій для роботи з одним екземпляром такого списку (у фігурних дужках параметри функції):

* push\_front, {value} - вставити новий вузол з заданим значенням на початок списку`,
* push\_back, {value} - вставити новий вузол з заданим значенням у кінець списку,
* size - підрахувати і повернути кількість вузлів списку
* clear - видалити всі вузли списку звільнивши всю виділену пам'ять.
* count\_neg - підрахувати і повернути кількість всіх від'ємних значень списку
* min - повернути найменше зі значень списку (NAN якщо список пустий)

Додатково реалізувати функцію print для виводу елементів списку.

**Тексти коду програм**

|  |
| --- |
| **main.c** |
| #include <ctype.h>  #include <float.h>  #include <math.h>  #include <progbase.h>  #include <progbase/console.h>  #include <stdbool.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <time.h>  int conReadLine(char str[], int maxBufLen) {  fgets(str, maxBufLen, stdin);  int bufLength = strlen(str);  if (str[bufLength - 1] == '\n') {  str[bufLength - 1] = '\0';  bufLength -= 1;  } else {  for (char ch; (ch = getchar()) != '\n';) {  }  }  return bufLength;  }  struct SLNode {  float data;  struct SLNode \*next;  };  struct DynArray {  int \*array;  int capacity;  int length;  };  void waiting() {  char key = 0;  printf("\nPress ANY KEY to return to commands list or to continue TEST CASE");  do {  key = Console\_getChar();  } while (key == 0);  Console\_clear();  }  void error(int n) {  Console\_clear();  puts("ERROR\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  switch (n) {  case 1:  printf(" You must initialize ARRAY before using it");  break;  case 2:  printf("Your input is WRONG or NOT NUMBER");  break;  case 3:  printf("WRONG POSITION");  break;  case 4:  printf("WRONG DOT POSITION");  break;  case 5:  printf("TOO MUCH DOTS");  break;  }  printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  waiting();  }  void dynarray\_init(struct DynArray \*pdarr) {  Console\_clear();  printf("DYNARRAY INIT\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  pdarr->capacity = 5;  pdarr->length = 0;  pdarr->array = malloc(pdarr->capacity \* sizeof(int));  if (pdarr->array == NULL) {  int r = rand() % 99 + 1;  printf("Cannot initialize ARRAY. Initializing with LENGTH = %i", r);  pdarr->length = r;  pdarr->array = malloc(pdarr->capacity \* sizeof(int));  }  printf("\n CAPACITY: %i\n\n LENGTH: %i\n\n ARRAY: ", pdarr->capacity,  pdarr->length);  for (int i = 0; i < pdarr->length; i++) {  printf("%i ", pdarr->array[i]);  }  printf("\n\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  waiting();  }  void dynarray\_deinit(struct DynArray \*pdarr) {  Console\_clear();  printf("DYNARRAY DEINIT\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  pdarr->length = 0;  pdarr->capacity = 0;  free(pdarr->array);  printf("\nARRAY has been DEINITialized\n\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  waiting();  }  void dynarray\_clear(struct DynArray \*pdarr) {  Console\_clear();  printf("DYNARRAY CLEAR\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  pdarr->length = 0;  printf("\n Now LENGTH became 0\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  waiting();  }  void dynarray\_reserve(struct DynArray \*pdarr, long int val) {  Console\_clear();  printf("DYNARRAY RESERVE\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  if (pdarr->capacity < val) {  pdarr->capacity = val;  int \*NewArray = realloc(pdarr->array, pdarr->capacity \* sizeof(int));  pdarr->array = NewArray;  printf("\n New CAPACITY is %i", val);  } else {  printf("\n New CAPACITY lower than PREVIOUS");  }  printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  waiting();  }  void dynarray\_push\_back(struct DynArray \*pdarr, int val) {  Console\_clear();  printf("DYNARRAY PUSH BACK\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  if (pdarr->capacity <= pdarr->length + 1) {  pdarr->capacity = pdarr->capacity + 1;  int \*NewArray = realloc(pdarr->array, pdarr->capacity \* sizeof(int));  pdarr->array = NewArray;  }  pdarr->array[pdarr->length] = val;  pdarr->length++;  printf(" VALUE %i has been added. Now LENGTH is %i", val, pdarr->length);  printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  waiting();  }  void dynarray\_erase(struct DynArray \*pdarr, int index) {  printf("DYNARRAY ERASE\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  if (index >= pdarr->length || index < 0) {  error(3);  return;  } else {  for (int i = index; i < pdarr->length; i++) {  pdarr->array[i] = pdarr->array[i + 1];  }  pdarr->length--;  printf(" Element №%i has been erased", index);  }  printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  waiting();  }  void dynarray\_print(struct DynArray \*pdarr) {  Console\_clear();  printf("DYNARRAY PRINT\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  printf("\n CAPACITY: %i\n\n LENGTH: %i\n\n ARRAY: ", pdarr->capacity,  pdarr->length);  for (int i = 0; i < pdarr->length; i++) {  printf("%i ", pdarr->array[i]);  }  printf("\n\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  waiting();  }  void dynarray\_test(struct DynArray \*pdarr) {  Console\_clear();  printf("DYNARRAY TEST\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n TEST CASE is: \n [1] "  "dynarray\_init\n [2] dynarray\_push\_back 0-9 \n [3] dynarray\_print\n "  " [4] dynarray\_reserve 100\n [4] dynarray\_erase 9-0\n [5] "  "dynarray\_print\n [6] dynarray\_clear\n [7] "  "dynarray\_deinit\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\nTap 1 to BEGIN or ENTER to abort "  "operation");  char key;  do {  key = Console\_getChar();  if (key == '\n') {  Console\_clear();  return;  }  } while (key != '1');  dynarray\_init(pdarr);  for (int i = 0; i < 10; i++) {  dynarray\_push\_back(pdarr, i);  }  dynarray\_print(pdarr);  dynarray\_reserve(pdarr, 100);  dynarray\_print(pdarr);  for (size\_t i = 0; i < 10; i++) {  dynarray\_erase(pdarr, 9 - i);  }  dynarray\_print(pdarr);  dynarray\_clear(pdarr);  dynarray\_deinit(pdarr);  }  void dynarray\_main() {  char \*\*trash;  Console\_clear();  char key;  int z = 0, q;  int isinit = 0;  struct DynArray dynarr;  do {  if (z == 0) {  printf("DYNARRAY MENU\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n [1] INIT\n [2] DEINIT\n "  "[3] CLEAR\n [4] RESERVE\n [5] PUSH\_BACK\n [6] ERASE\n [7] "  "PRINT\n [8] TEST CASE\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  } else if (z == 1) {  printf("CURRENT CAPACITY: %i | LENGTH: %i | ARRAY: ", dynarr.capacity,  dynarr.length);  for (int i = 0; i < dynarr.length; i++) {  printf("%i ", dynarr.array[i]);  }  printf(" |\n\nDYNARRAY MENU\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n [1] INIT\n [2] "  "DEINIT\n [3] CLEAR\n [4] RESERVE\n [5] PUSH\_BACK\n [6] "  "ERASE\n [7] PRINT\n [8] TEST CASE\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  z = 2;  }  key = Console\_getChar();  switch (key) {  case '1':  if (isinit == 1) {  dynarr.length = 0;  dynarr.capacity = 0;  free(dynarr.array);  }  dynarray\_init(&dynarr);  isinit = 1;  z = 1;  break;  case '2':  if (z == 0) {  error(1);  } else {  dynarray\_deinit(&dynarr);  isinit = 0;  z = 0;  }  break;  case '3':  if (z == 0) {  error(1);  } else {  dynarray\_clear(&dynarr);  z = 1;  }  break;  case '4':  q = 1;  if (z == 0) {  error(1);  } else {  Console\_clear();  char str[1000];  printf("Enter capacity to RESERVE: ");  int len = conReadLine(str, 1000);  for (int i = 0; i < len; i++) {  if (isdigit(str[i]) == 0) {  error(2);  z = 1;  q = 0;  break;  }  }  if (q != 0) {  dynarray\_reserve(&dynarr, strtol(str, trash, 10));  }  z = 1;  }  break;  case '5':  if (z == 0) {  error(1);  } else {  Console\_clear();  char str[1000];  printf("Enter VALUE to add: ");  int len = conReadLine(str, 1000);  for (int i = 0; i < len; i++) {  if (isdigit(str[i]) == 0) {  error(2);  z = 1;  break;  }  }  dynarray\_push\_back(&dynarr, strtol(str, trash, 10));  z = 1;  break;  }  break;  case '6':  if (z == 0) {  error(1);  } else {  Console\_clear();  char str[1000];  printf("Enter POSITION to ERASE: ");  int len = conReadLine(str, 1000);  for (int i = 0; i < len; i++) {  if (isdigit(str[i]) == 0) {  error(2);  z = 1;  break;  }  }  dynarray\_erase(&dynarr, strtol(str, trash, 10));  z = 1;  break;  }  break;  z = 1;  break;  case '7':  if (z == 0) {  error(1);  } else {  dynarray\_print(&dynarr);  z = 1;  }  break;  case '8':  dynarray\_test(&dynarr);  z = 0;  break;  }  } while (key != '\n');  if (isinit == 1) {  dynarray\_deinit(&dynarr);  }  }  void slnode\_clear(struct SLNode \*head) {  struct SLNode \*node = head;  while (node != NULL) {  struct SLNode \*next = node->next;  free(node);  node = next;  next = NULL;  }  }  struct SLNode \*slnode\_create(struct SLNode \*head, float data) {  struct SLNode \*pnode = malloc(sizeof(struct SLNode));  if (pnode == NULL) {  printf("Memmory error\n");  slnode\_clear(head);  exit(1);  } else {  pnode->data = data;  pnode->next = NULL;  return pnode;  }  }  struct SLNode \*slnode\_push\_front(struct SLNode \*head, float value) {  struct SLNode \*new\_node = slnode\_create(head, value);  new\_node->next = head;  return new\_node;  }  struct SLNode \*slnode\_push\_back(struct SLNode \*head, float value) {  struct SLNode \*new\_node = slnode\_create(head, value);  if (head == NULL) {  return new\_node;  } else {  struct SLNode \*node = head;  while (node->next != NULL)  node = node->next;  node->next = new\_node;  return head;  }  }  size\_t slnode\_size(struct SLNode \*head) {  size\_t size = 0;  struct SLNode \*node = head;  while (node != NULL) {  size++;  node = node->next;  }  return size;  }  size\_t slnode\_count\_neg(struct SLNode \*head) {  size\_t cnt = 0;  struct SLNode \*node = head;  while (node != NULL) {  if (node->data < 0)  cnt++;  node = node->next;  }  return cnt;  }  float slnode\_min(struct SLNode \*head) {  struct SLNode \*node = head;  float min = node->data;  if (node == NULL) {  return NAN;  }  while (node != NULL) {  if (node->data <= min) {  min = node->data;  }  node = node->next;  }  return min;  }  void slnode\_print(struct SLNode \*head) {  struct SLNode \*node = head;  printf("\n NODE: ");  while (node != NULL) {  printf("[%.3f] ", node->data);  if (node->next != NULL) {  printf("-> ");  }  node = node->next;  }  }  void slnode\_main() {  char \*\*trash;  Console\_clear();  char key;  int z = 0;  struct SLNode \*head = NULL;  int isinit = 1;  do {  if (z != 1) {  if (z == 2) {  struct SLNode \*node = head;  printf("\n NODE: ");  while (node != NULL) {  printf("[%.3f] ", node->data);  if (node->next != NULL) {  printf("-> ");  }  node = node->next;  }  }  printf(  "\n\nSLNODE MENU\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n [1] PUSH\_FRONT\n [2] "  "PUSH\_BACK\n [3] SIZE\n [4] CLEAR\n [5] COUNT\_NEG\n [6] MIN\n "  "[7] PRINT\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  z = 1;  }  key = Console\_getChar();  int dotc = 0;  float a;  switch (key) {  case '\n':  break;  case '1':  Console\_clear();  printf("Enter value to PUSH\_FRONT: ");  scanf("%f", &a);  head = slnode\_push\_front(head, a);  Console\_clear();  z = 2;  break;  case '2':  Console\_clear();  printf("Enter value to PUSH\_BACK: ");  scanf("%f", &a);  head = slnode\_push\_back(head, a);  Console\_clear();  z = 2;  break;  case '3':  Console\_clear();  printf("SLNODE SIZE\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  printf("\n SIZE: %zu", slnode\_size(head));  printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  waiting();  z = 2;  break;  case '4':  Console\_clear();  printf("SLNODE CLEAR\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  slnode\_clear(head);  printf("\nNow SLNODE empty\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  waiting();  key = '\n';  z = 2;  break;  case '5':  Console\_clear();  printf("SLNODE COUNT\_NEG\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  printf("\n NEGATIVE COUNT: %zu\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n",  slnode\_count\_neg(head));  waiting();  z = 2;  break;  case '6':  Console\_clear();  printf("SLNODE MIN\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  if (slnode\_size(head) == 0) {  puts("\n NAN\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  } else {  printf("\n MINIMAL: %f\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n", slnode\_min(head));  }  waiting();  z = 2;  break;  case '7':  Console\_clear();  printf("SLNODE PRINT\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  slnode\_print(head);  printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  waiting();  z = 2;  break;  }  } while (key != '\n');  if (isinit == 1)  {  free(head);  }  }  void main(void) {  srand(time(0));  Console\_clear();  char key;  int z = 1;  do {  if (z != 0) {  Console\_clear();  puts("MAIN MENU\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n [1] DYNARRAY MENU\n [2] SLNODE "  "MENU\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  z = 0;  }  key = Console\_getChar();  switch (key) {  case '1':  dynarray\_main();  z = 1;  break;  case '2':  slnode\_main();  z = 1;  break;  }  } while (key != '\n');  Console\_clear();  } |

**Приклади результатів**

Приклади виконання з консолі:

DYNARRAY INIT

|  |
| --- |
| DYNARRAY INIT  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  CAPACITY: 5  LENGTH: 0  ARRAY:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Press ANY KEY to return to commands list or to continue TEST CASE |

DYNARRAY DEINIT

|  |
| --- |
| DYNARRAY DEINIT  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  ARRAY has been DEINITialized  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Press ANY KEY to return to commands list or to continue TEST CASE |

DYNARRAY CLEAR

|  |
| --- |
| DYNARRAY CLEAR  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Now LENGTH became 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Press ANY KEY to return to commands list or to continue TEST CASE |

DYNARRAY RESERVE

|  |
| --- |
| Enter capacity to RESERVE: 100  DYNARRAY RESERVE  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  New CAPACITY is 100  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Press ANY KEY to return to commands list or to continue TEST CASE |

DYNARRAY PUSH BACK

|  |
| --- |
| Enter VALUE to add: 100  DYNARRAY PUSH BACK  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  VALUE 100 has been added. Now LENGTH is 1  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Press ANY KEY to return to commands list or to continue TEST CASE |

DYNARRAY ERASE

|  |
| --- |
| Enter POSITION to ERASE: 1  DYNARRAY ERASE  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Element №1 has been erased  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Press ANY KEY to return to commands list or to continue TEST CASE |

DYNARRAY PRINT

|  |
| --- |
| DYNARRAY PRINT  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  CAPACITY: 100  LENGTH: 1  ARRAY: 100  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Press ANY KEY to return to commands list or to continue TEST CASE |

SLNODE PUSH FRONT

|  |
| --- |
| Enter value to PUSH\_FRONT: 100 |

SLNODE PUSH BACK

|  |
| --- |
| Enter value to PUSH\_BACK: 100 |

SLNODE SIZE

|  |
| --- |
| SLNODE SIZE  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  SIZE: 2  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Press ANY KEY to return to commands list or to continue TEST CASE |

SLNODE CLEAR

|  |
| --- |
| SLNODE CLEAR  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Now SLNODE empty  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Press ANY KEY to return to commands list or to continue TEST CASE |

SLNODE COUNT NEG

|  |
| --- |
| SLNODE COUNT\_NEG  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  NEGATIVE COUNT: 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Press ANY KEY to return to commands list or to continue TEST CASE |

SLNODE MIN

|  |
| --- |
| SLNODE MIN  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  NAN  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Press ANY KEY to return to commands list or to continue TEST CASE |

SLNODE PRINT

|  |
| --- |
| SLNODE PRINT  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  NODE:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Press ANY KEY to return to commands list or to continue TEST CASE |

VALGRIND РЕЗУЛЬТАТИ

|  |
| --- |
| ==11101==  ==11101== HEAP SUMMARY:  ==11101== in use at exit: 0 bytes in 0 blocks  ==11101== total heap usage: 5 allocs, 6 frees, 2,100 bytes allocated  ==11101==  ==11101== All heap blocks were freed -- no leaks are possible  ==11101==  ==11101== For lists of detected and suppressed errors, rerun with: -s  ==11101== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0) |

**Висновки**

Виконавши дану лабораторну роботу я навчився використовувати динамічні списки на основі динамічного масиву і динамічні списки зв’язаних вузлів. Компіляція всього коду відбувалася за допомогою компілятора gcc.