Московский Авиационный Институт

(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Лабораторная работа № 3
по курсу «Операционные системы»

Студент:	Голубев В.С.
Группа:	М8О-206Б-18
Вариант:	7
Преподаватель:	Миронов Е.С.
Оценка:	
Дата:	

Москва, 2019

1. Постановка задачи

Требуется создать динамическую библиотеку, которая реализует определенный функционал – работу с массивом, содержащим целые 32-битные числа.

2. Описание работы программы

В файле APIarray.c реализованы функции для работы с массивом : arrayCreate, arrayGet, arrayInsert, arrayDelete, arrayResize, arrayDestroy, arayPrint. В файле dynamic.c эти функци загружаются в память, выделенную для программы

3.Листинг программы

APIarray.c

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

#include "APIarray.h"

array* arrayCreate(size_t size) {
if(size <= 0) {
array* a = (array*)malloc(sizeof(array));
a->data = (int*)malloc(sizeof(int));
a->size = 1;
}
array* a = (array*)malloc(sizeof(array));
a->data = (int*)malloc(sizeof(int) * size);
a->size = size;
return a;
}

void arrayInsert(array* a, size_t pos, int val) {
if(pos >= a->size) {
printf("Out of bounds\n");
exit(-1);
}
a->data[pos] = val;
```

```
}

int arrayGet(array* a, size_t pos) {
if(pos >= a->size) {
printf("Out of bounds\n");
exit(-1);
}
return a->data[pos];
}

void arrayDelete(array* a, size_t pos) {
if(pos >= a->size) {
printf("Out of bounds\n");
exit(-1);
}
int* newData = (int*)malloc(sizeof(int) * (a->size - 1));
for(unsigned i = 0; i < a->size; ++i) {
if(i < pos) {
newData[i] = a->data[i];
} else if(i > pos) {
newData[i - 1] = a->data[i];
}
}
free(a->data);
a->data = newData;
}

void arrayResize(array* a, size_t size) {
int* newData = (int*)malloc(sizeof(int) * size);
if(size <= a->size) {
for(unsigned i = 0; i < size; ++i) {
newData[i] = a->data[i];
}
} else {
for(unsigned i = 0; i < a->size; ++i) {
newData[i] = a->data[i];
```

```
}
}
free(a->data);
a->data = newData;
a->size = size;
}

void arrayDestroy(array* a) {
free(a->data);
a->size = 0;
free(a);
}

void arrayPrint(array* a) {
for(unsigned i = 0; i < a->size; ++i) {
printf("%d ", a->data[i]);
}
printf("\n");
}
```

APIarray.h

```
#ifndef _ARRAY_API_H
#define _ARRAY_API_H

#include <stdlib.h>

typedef struct array array;

struct array{
int* data;
size_t size;
};

array* arrayCreate(size_t size);
void arrayInsert(array* a, size_t pos, int val);
int arrayGet(array* a, size_t pos);
```

```
void arrayDelete(array* a, size_t pos);
void arrayResize(array* a, size_t size);
void arrayDestroy(array* a);
void arrayPrint(array* a);
#endif // _ARRAY_API_H
```

Static.c

```
#include <stdio.h>

#include "APIarray.h"

int main() {
int command = 0;
array* a;
int value = 0;
int pos = 0;
printf("1 - create array with given size\n");
printf("2 - insert element to array at given position\n");
printf("3 - get element value on given position\n");
printf("4 - delete element from given position\n");
printf("5 - resize array to given size\n");
printf("6 - print array\n");
printf("0 - exit\n");
while(scanf("%d", &command) && command) {
if(command == 1) {
printf("Enter value\n");
scanf("%d", &value);
a = arrayCreate(value);
} else if(command == 2) {
printf("Enter position and value\n");
scanf("%d %d", &pos, &value);
arrayInsert(a, pos, value);
} else if(command == 3) {
printf("Enter position\n");
scanf("%d", &pos);
printf("%d\n", arrayGet(a, pos));
```

```
} else if(command == 4) {
printf("Enter position\n");
scanf("%d", &pos);
arrayDelete(a, pos);
} else if(command == 5) {
printf("Enter value\n");
scanf("%d", &value);
arrayResize(a, value);
} else if(command == 6) {
arrayPrint(a);
}
}
arrayDestroy(a);
return 0;
}
```

dynamic.c

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

#include "APIarray.h"

int main() {
array* (*arrayCreate)(size_t size);
void (*arrayInsert)(array* a, size_t pos, int val);
int (*arrayGet)(array* a, size_t pos);
void (*arrayDelete)(array* a, size_t pos);
void (*arrayResize)(array* a, size_t size);
void (*arrayDestroy)(array* a);
void (*arrayPrint)(array* a);
char* error;
void* libHandle;
libHandle = dlopen("./libarr.so", RTLD_LAZY);
if(!libHandle) {
printf("%s\n", dlerror());
```

```
return -1;
}
arrayCreate = dlsym(libHandle, "arrayCreate");
arrayInsert = dlsym(libHandle, "arrayInsert");
arrayGet = dlsym(libHandle, "arrayGet");
arrayDelete = dlsym(libHandle, "arrayDelete");
arrayResize = dlsym(libHandle, "arrayResize");
arrayDestroy = dlsym(libHandle, "arrayDestroy");
arrayPrint = dlsym(libHandle, "arrayPrint");
printf("1 - create array with given size\n");
printf("2 - insert element to array at given position\n");
printf("3 - get element value on given position\n");
printf("4 - delete element from given position\n");
printf("5 - resize array to given size\n");
printf("6 - print array\n");
printf("0 - exit\n");
int command = 0;
array* a;
int value = 0;
int pos = 0;
while(scanf("%d", &command) && command) {
if(command == 1) {
printf("Enter value\n");
scanf("%d", &value);
a = (*arrayCreate)(value);
} else if(command == 2) {
printf("Enter position & value\n");
scanf("%d %d", &pos, &value);
(*arrayInsert)(a, pos, value);
} else if(command == 3) {
printf("Enter position\n");
scanf("%d", &pos);
printf("%d\n", (*arrayGet)(a, pos));
} else if(command == 4) {
printf("Enter position\n");
scanf("%d", &pos);
```

```
(*arrayDelete)(a, pos);
} else if(command == 5) {
printf("Enter value\n");
scanf("%d", &value);
(*arrayResize)(a, value);
} else if(command == 6) {
(*arrayPrint)(a);
}
}
(*arrayDestroy)(a);
return 0;
}
```

makefile

```
all: main

main: libarr.so dynamic.o static.o
gcc -o static static.o -L. -larr -Wl,-rpath,.
gcc -o dynamic dynamic.o -ldl

dynamic.o: dynamic.c
gcc -c dynamic.c

static.o: static.c
gcc -c static.c

libarr.so: APIarray.o
gcc -shared -o libarr.so APIarray.o

arrayAPI.o : APIarray.c
gcc -c -fPIC APIarray.c

clean:
rm -f *.o *.so static dynamic
```

4.Вывод

В ходе выполнения данной ЛР я познакомился с динамическими библиотеками, которые повсеместно используются в современных программах, позволяя ускорить работу со сторонними библиотеками, т.к. включают в программу только необходимые функции, а не все подряд, указанные в библиотеке.