МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

**«Вятский государственный университет»**

**(ФГБОУ ВО «ВятГУ»)**

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Исследование состояния динамической памяти

Отчет

Лабораторная работа №2 по дисциплине

«Технологии программирования»

Выполнил студент группы ИВТ-21 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Кудяшев Я.Ю./

Проверил преподаватель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Долженкова М.Л./

Киров 2021

**Задание**

Разработать приложение, реализующее динамическую структуру – deque, каждый элемент которого string и число типа double. Изучить дамп памяти при вставке и удалении элементов.

**Листинг программы**

#include <iostream>

#include <string>

using namespace std;

struct Types //Using types

{

double doub;

string str;

};

struct Point //Pointers

{

Test value;

Point\* prev;

Point\* next;

};

struct Deque //Deque

{

Point\* start;

Point\* end;

int size;

};

int\* DequeNew(Deque\* deque) //New deque

{

deque->start = NULL;

deque->end = NULL;

deque->size = 0;

return 0;

}

int\* DequeDelete(Deque\* deque) //Deleting of structure

{

while (deque->start != NULL)

{

Point\* next = deque->start->next;

delete deque->start;

deque->start = next;

}

deque->start = deque->end = NULL;

deque->size = 0;

return 0;

}

void DequePushBack(Deque\* deque, Types value) //Adding to the end

{

Point\* temp = new Point;

temp->value = value;

temp->next = temp->prev = NULL;

if (deque->start == NULL) {

deque->end = deque->start = temp;

}

else

{

deque->end->next = temp;

temp->prev = deque->end;

deque->end = temp;

}

++deque->size;

}

int DequePopBack(Deque\* deque) //Deleting from the end

{

if (deque->size == 0)

{

cout << "Deque is empty.\n";

return -1;

}

else

{

Point\* prev = deque->end->prev;

delete deque->end;

deque->end = prev;

if (deque->end == NULL)

deque->start = deque->end;

else

deque->end->next = NULL;

-deque->size;

return 0;

}

}

int DequeSize(Deque\* deque) //Size of the deque

{

if (deque->size == 0)

{

cout << "Deque is empty.\n";

return -1;

}

else

{

int s = deque->size;

cout << "Size of the deque is " << s << endl;

return s;

}

}

int DequePushHead(Deque\* deque, Types value) //Adding to the begining

{

Point\* temp = new Point;

temp->value = value;

temp->next = temp->prev = NULL;

if (deque->start == NULL) {

deque->end = deque->start = temp;

}

else {

deque->start->prev = temp;

temp->next = deque->start;

deque->start = temp;

}

++deque->size;

return 0;

}

int DequePopHead(Deque\* deque) //Deleting from the begining

{

if (deque->size == 0)

{

cout << "Deque is empty.\n";

return -1;

}

else

{

Point\* next = deque->start->next;

delete deque->start;

deque->start = next;

if (deque->start == NULL)

deque->end = deque->start;

else

deque->start->prev = NULL;

-deque->size;

return 0;

}

}

int\* DequePrintStart(Deque\* deque) //Show first element of the structure

{

Point\* temp;

cout << "First double-number is " << deque->start->value.doub << endl;

cout << "First string is " << deque->start->value.str << endl;

return 0;

}

int\* DequePrintEnd(Deque\* deque) //Show last element of the structure

{

Point\* temp;

cout << "Last double-number is " << deque->end->value.doub << endl;

cout << "Last string is " << deque->end->value.str << endl;

return 0;

}

int main()

{

Types result;

int variant;

Deque deque;

DequeNew(&deque);

bool gg = false;

int menu;

while (gg == false) {

system("cls");

cout << "Deque for working with data types like string and double.\n"<<endl;

cout << "1. Add an element to the begining.\n";

cout << "2. Add an element to the end.\n";

cout << "3. Delete an element from the begining.\n";

cout << "4. Delete an element from the end.\n";

cout << "5. Show the first element of the deque.\n";

cout << "6. Show the last element of the deque.\n";

cout << "7. Size of the deque.\n";

cout << "8. Clear the structure.\n";

cout << "9. Exit.\n";

cout << endl;

cin >> menu;

switch (menu)

{

case 1: {

cout << "Enter a double-number: ";

cin >> result.doub;

cout << endl;

cout << "Enter a string: ";

cin >> result.str;

cout << endl;

DequePushHead(&deque, result);

break;

}

case 2: {

cout << "Enter a double-number: ";

cin >> result.doub;

cout << endl;

cout << "Enter a string: ";

cin >> result.str;

cout << endl;

DequePushBack(&deque, result);

break;

}

case 3: {

DequePopHead(&deque);

system("pause");

break;

}

case 4: {

DequePopBack(&deque);

system("pause");

break;

}

case 5: {

DequePrintStart(&deque);

system("pause");

break;

}

case 6: {

DequePrintEnd(&deque);

system("pause");

break;

}

case 7: {

DequeSize(&deque);

system("pause");

break;

}

case 8: {

DequeDelete(&deque);

break;

}

case 9: {

cout << "Bye-bye\n";

gg = true;

system("pause");

break;

}

default: {

cout << "Try again" << endl;

system("pause");

break;

}

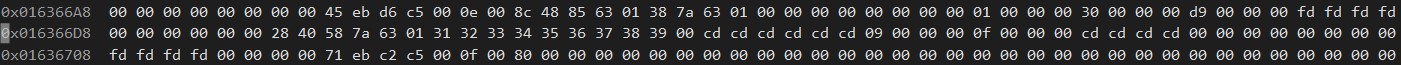
}

}

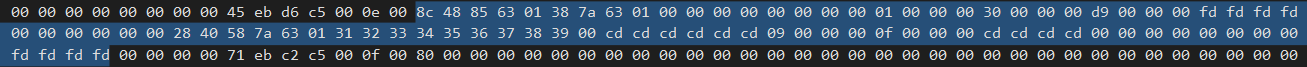
}

**Инструкция по дамбу памяти**

Имеем структуру данных Deque для типов double и string. Добавим в начало число 12 типа double и строку “123456789”. В результате имеем дамб памяти вида:



Наша структура данных, вместе с пользовательской, выделена синим цветом:



Расшифровка всего, что написано в блоке памяти (цвета могут повторяться):

8c 48 85 63 01 38 7a 63 01 00 00 00 00 00 00 00 00 01 00 00 00 30 00 00 00 d9 00 00 00 fd fd fd fd 00 00 00 00 00 00 28 40 58 7a 63 01 31 32 33 34 35 36 37 38 39 00 cd cd cd cd cd cd 09 00 00 00 0f 00 00 00 cd cd cd cd 00 00 00 00 00 00 00 00 fd fd fd fd

8c – использование поля. Любое значение (не равное 80) – используется, 80 – не используется.

48 85 63 01 – ссылка на предыдущий занятый блок памяти.

38 7a 63 01 – ссылка на следующий занятый блок памяти.

00 00 00 00 – ссылка на имя файла подкачки.

00 00 00 00 – количество блоков памяти, если данный блок фрагментирован.

01 00 00 00 – количество обращений к блоку памяти.

30 00 00 00 – размер блока структуры (в данном случае 3016 = 4810).

d9 00 00 00 – количество используемых блоков памяти.

fd fd fd fd – флаг начала/конца пользовательской структуры.

00 00 00 00 00 00 28 40 – структура, которую помещают в deque (в данном случае 8 байт под тип double, в которые записано число 12 в шестнадцатеричной системе счисления в обратном порядке).

58 7a 63 01 – ссылка на блок памяти, где хранятся данные нашей строки.

31 32 33 34 35 36 37 38 39 00 cd cd cd cd cd cd – структура, которую помещают в deque (в данном случае 16 байт под тип string, в которые записана строка “123456789” в шестнадцатеричной системе счисления. Под любую строку изначально выделяется 16 байт памяти. В данном случае cd является пустым элементом памяти, т.к. строка длиной 10 байт, а не 16).

09 00 00 00 – количество символом/элементов в строке.

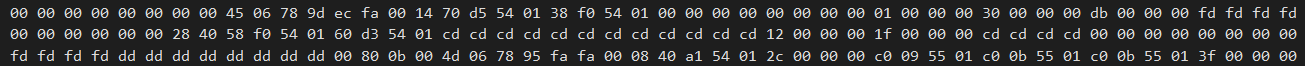
0f 00 00 00 – количество памяти, выделенное под хранение строки.

cd cd cd cd – дополнение блока до числа, кратного 4 (программа запускалась на 32-битной системе. 32 / 8 = 4, где 8 – количество бит в байте. 4 байта – размер машинного слова).

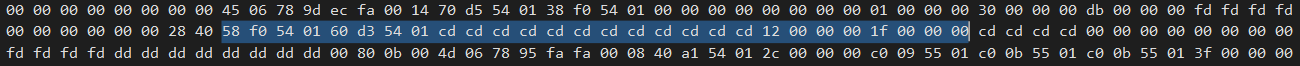
00 00 00 00 – адрес предыдущего элемента.

00 00 00 00 – адрес следующего элемента.

В случае, если длина строки более 16 символов. В данном случае в строку записано “123456789123456789”:



Нас интересует данный отрезок памяти:



Расшифровка данного блока памяти:

58 f0 54 01 60 d3 54 01 cd cd cd cd cd cd cd cd cd cd cd cd 12 00 00 00 1f 00 00 00

58 f0 54 01 – ссылка на блок памяти, где хранятся данные нашей строки.

60 d3 54 01 cd cd cd cd cd cd cd cd cd cd cd cd – ссылка на блок памяти, где хранится наша строка (т.к. длина строки превысила 16 символов, то создалась ссылка на новый блок памяти).

12 00 00 00 – количество символов в строке (1216 = 1810).

1f 00 00 00 – количество памяти, выделенное под хранение строки (в данном случае оно равно 16 \* 2 = 32, т.к. количество символов превысило 16. Каждый раз, при нехватке памяти, она увеличивается на 16).

При переходе по данной ссылке в блоке памяти можно увидеть нашу структуру string:



**Вывод**

В ходе выполнения лабораторной работы были закреплены навыки работы с языком C++.

Главной особенностью данной лабораторной работы является необходимость изучения дамба памяти при вставке и удалении элемента. Увидеть что и как хранится в памяти можно при помощи специальных возможностей Visual Studio. Открыть вкладку отображения памяти можно нажав «Отладка > Окна > Память», поставив ранее точку останова в программе. После проделанных действий открывается окно с большим количеством непонятной, на первый взгляд, информации, часть которой предстоит изучить. Поначалу неподготовленный пользователь может растеряться, однако после более тщательного изучения содержимого памяти всё становится намного понятнее.

Хочется отметить, что в памяти данные хранятся в шестнадцатеричной системе счисления и задом наперёд, т.е. считать байты нужно с конца в начало. После занесения данных в структуру создаётся блок памяти, состоящий из пользовательских данных и специальных ссылок. Первые четыре байта отведены под ссылку на предыдущий занятый блок памяти, вторые четыре байта под ссылку на следующий занятый блок памяти, третьи четыре байта под ссылку на имя файла подкачки, четвёртые четыре байта под количество блоков памяти, если данный блок фрагментирован, пятые четыре байта под количество обращений к блоку памяти, шестые четыре байта под размер блока структуры и седьмые четыре байта под количество используемых блоков памяти. После этих данных начинается пользовательская структура для хранения элементов типа double и string. Она заключена между флагами начала и конца, представляющими из себя 4 байта «fdfdfdfd». Первые 8 байт отведены под тип double, т.к. он фиксировано занимает 8 байт. В этих 8 байтах и хранится число, введённое пользователем. Далее, после этих 8 байт, выделяется память под тип string. В отличие от double string является динамической структурой и заранее не известно количество памяти, необходимое для её хранения. Из-за данной особенности изначально выделяется 16 байт под тип string. Если же длина введённой пользователем строки превысит 16 символов, то количество выделенной памяти увеличится в два раза, т.е. 16\*2=32 и так далее. После увеличения памяти в области для записи данных будет хранится четырёхбайтный адрес области, в которой теперь хранится структура string. Максимальное количество памяти, которое может быть выделено под тип string – 2 ГБ. Это связано с тем, что под каждое приложение изначально выделяется 2 ГБ памяти. После данных string идут 4 байта, в которых хранится размер string, т.е. количество символов. И, наконец, последние 8 байт отведены под адреса предыдущего и следующего элементов структуры. Кроме того, размер пользовательского блока всегда будет кратен числу n/8, где n – это разрядность системы. Если же блок не удовлетворяет данным требованиям, то происходит его дополнение до числа, кратного 4, т.к. размер машинного слова = 4 байт.