Лабораторная работа №1

Аллокатор памяти общего назначения

Реализация аллокатора памяти общего назначения основана на двусвязном списке: каждый блок содержит ссылку на предыдущий и следующий.

Память, над которой прозводит операции аллокатор делится на блоки. Каждый блок имеет хедер, который содержит информацию о данном блоке.

is_available (1 byte) - проверка, занят ли блок

size (2 byte) - размер блока

previous_size (2 byte) - размер предыдущего блока

По причине выравнивания данных размер хедера составляет 8 байтов.

Реализация

При создании аллокатор запрашивает кусок памяти, используя стандартный аллокатор C++.

mem_alloc(size_t size)

Выделение области памяти размером size. Если аллокатор не может выделить блок запрашиваемого размера, то возвращается nullptr.

Выделенная память должна быть выровнена. В данной реализации каждый блок выровнен по 4 байта. Поиск доступного блока производится с помощью метода первого походящего (поиск завершается, когда найден первый подходящий блок). Когда блок найден, он разделяется на два: один - для данных, а другой - для последующего выделения памяти.

mem_free(void* addr)

Освобождение выделенного по адресу addr блока памяти.

Полученный функцией адрес проверяется во избежание освобождения неправильного блока. После освобождения блока аллокатор пытается произвести дефрагментацию памяти путем его слияния с соседними блоками, если те свободны.

mem_free()

Освобождение всей выделенной памяти производится путем слияния всех блоков в один.

mem_realloc(void* addr, size_t size)

Перераспределение блока памяти по адресу addr с новым размером size.

Для перераспределения блока памяти мы сначала проверяем адрес и проверяем блок на возможность осуществления реаллокации памяти. Если места не хватает даже после слияния со свободными соседними блоками, аллокатор пытается выделить память, используя mem_alloc(size_t size).

Примеры

Код (аллокация) :

```
allocator.mem_alloc(512);
allocator.mem_alloc(256);
allocator.mem_alloc(128);
allocator.mem_alloc(64);
allocator.mem_alloc(32);
allocator.mem_alloc(16);
allocator.mem_alloc(8);
allocator.mem_dump();
```

Результат:

```
Total size: 1024
Used: 8
Header size: 8
+ 015EE6D0 1016

Total size: 1024
Used: 1024
Header size: 8
- 015EE6D0 512
- 015EE8D8 256
- 015EE9E0 128
- 015EEA68 64
- 015EEAB0 16
+ 015EEAC8 0
```

Код (освобождение):

```
auto* addr1 = allocator.mem_alloc(512);
auto* addr2 = allocator.mem_alloc(256);
auto* addr3 = allocator.mem_alloc(128);
allocator.mem_dump();

allocator.mem_free(addr1);
allocator.mem_dump();

allocator.mem_free(addr2);
allocator.mem_dump();

allocator.mem_free(addr3);
allocator.mem_dump();

allocator.mem_alloc(500);
allocator.mem_alloc(500);
allocator.mem_dump();

allocator.mem_free();
allocator.mem_dump();
```

Результат:

```
Total size: 1024
Used: 928
Header size: 8
- 00F7E6E0 512
- 00F7E8E8 256
- 00F7E9F0 128
+ 00F7EA78 96

Total size: 1024
Used: 416
Header size: 8
+ 00F7E6E0 512
- 00F7E8E8 256
- 00F7E9F0 128
+ 00F7EA78 96

Total size: 1024
Used: 152
Header size: 8
+ 00F7E6E0 776
- 00F7E9F0 128
+ 00F7EA78 96

Total size: 1024
Used: 8
Header size: 8
+ 00F7E6E0 776
+ 00F7E9F0 128
+ 00F7EA78 96

Total size: 1024
Used: 516
Header size: 8
- 00F7E6E0 500
+ 00F7E8DC 268
+ 00F7E9F0 128
+ 00F7EA78 96
```

Код(реаллокация):

```
auto* loc = allocator.mem_alloc(200);
allocator.mem_alloc(200);
allocator.mem_dump();

allocator.mem_realloc(loc, 20);
allocator.mem_dump();

allocator.mem_realloc(loc, 200);
allocator.mem_dump();

allocator.mem_realloc(loc, 500);
allocator.mem_dump();
```

Результат:

```
Total size: 1024
Used: 424
Header size: 8
- 00C8E6E0 200
- 00C8E7B0 200
+ 00C8E880 600

Total size: 1024
Used: 452
Header size: 8
- 00C8E6E0 20
+ 00C8E6FC 172
- 00C8E7B0 200
+ 00C8E880 600

Total size: 1024
Used: 644
Header size: 8
- 00C8E6E0 200
- 00C8E7B0 200
+ 00C8E880 600

Total size: 1024
Used: 952
Header size: 8
+ 00C8E6E0 200
- 00C8E7B0 200
- 00C8E880 500
+ 00C8EA7C 92
```