Профилирование памяти в IDE MS VisualStudio 2019

При отладке программ очень часто возникает необходимость в профилировании выделяемой памяти в программе. Иногда для этого используют специальные библиотеки для профилирования, такие как CRT (почитать можно тут). Если не нужна точная оценка выделяемой памяти, можно воспользоваться и функционалом стандартного средства отладки программы в MS VS.

	+⇒ X	* Ø	Обозреватель решений
profiler	 (Глобальная область) 	- +	G C 🔂 🗐 🐻 - 5 🗗 [
1 2 3 4	<pre>#include <iostream> constexpr auto BUFFER_SIZE = 1024 * 1024; using namespace std;</iostream></pre>	Ť	Обозреватель решений — поиси Решение "profiler" (проекты Profiler
5 6 7 8 9	<pre>void foo_bad() { int* a = new int[BUFFER_SIZE]; // TODO }</pre>		 ▶ ■■ Ссылки ▶ ■■ Ссылки ▶ ■В Нешние зависимос ∡ Псходные файлы ▶ *+ Source.cpp
10 11 12 13 14 15	<pre>void foo_good() { int* a = new int[BUFFER_SIZE]; // TOD0 delete[] a; }</pre>		≓ Файлы заголовков ≓ Файлы ресурсов
16 17 18 19 20	<pre>Dint main() { // make memory screenshot foo_good(); // make memory screenshot foo_bad(); // make memory screenshot</pre>		Обозреватель решен Измен

Для примера напишем такую программу:

В данном примере в функции foo_good() память выделяется и высвобождается, а в функции foo_bad() специально сделана ошибка, память не высвобождается, что приводит к утечке памяти.

Для профилирования поставим метку отладки на функции main() (16 строка) и выберем сборку проекта в режиме отладки (Debug). Далее запустим локальный отладчик (красная стрелка на предыдущем скриншоте) и в окне справа откроется меню профилирования. Если окно не открылось, вызовите его через пункт меню «Отладка - Окна - Показать средства диагностики» или комбинацией клавиш Ctrl+Alt+F2:

Sourceac	· · · · ·	Средства диагностики	· · · ^
💁 profile	er - (Глобальная ог - 🏵 main() - 🕇	🔅 🔁 🔍 🔍 📶	
6 7 8	<pre>int* a = new int[BUFFER_SIZE]; // TODO }</pre>	Сеанс диагностики: 0 мс	
9 10 11 12 13 14 15 0 16 17 18	<pre>void foo_good() { int* a = new int[BUFFER_SIZE]; // TOD0 delete[] a; } int main() { // make memory screenshot foo good();</pre>	 ▲ События II ▲ Память процесса 0 0 4 ЦП (% всех процессоров) 	100
19 20 21 22 23	<pre>// make memory screenshot foo_bad(); // make memory screenshot return EXIT_SUCCESS; }</pre>	100 Сводка События Использование памяти Использование ЦП	100
		🔯 Сделать снимок 💫 Просмотреть кучу	
		Х Удалить 🛱 Профилирова	ние кучи
		Идентификатор Время Выделения (разл.) Размер кучи (разн.)	
100 %	 Ф Проблемы не найдены. Стр: 16 Симв: 1 	Профилирование кучи со снимками отключено. Включить снимки, чтобы начать профилирование кучи для этого со отладки. Это снизит производительность отлаживаемого процесс	еанса са.

Найдите вкладку «использование памяти» И включите режим получения снимков памяти для профилирования кучи (при повторных запусках включать снимки не требуется).

В окне отладки у Вас в разделе «Использование памяти» появится кнопка «Сделать снимок»:

0			
Сводка События	Использование памяти	и Ист. породиние ЦП	
🖸 Сделать снимок	🔎 Просмотреть кучу	🗙 Удалить	
Идентифи тор Вр	емя Выделения (разл.)	Размер кучи (разн.)	

Под этой кнопкой находится таблица, которая сначала будет пустой и в

ней будет появляться новая строка, каждый раз, когда вы будете нажимать кнопку «Сделать снимок». Каждая запись отображает состояние памяти в данный момент программы.

Для отладки утечек сделаем снимок памяти в начале программы (данные о начальном размере кучи могут отличаться от приведённых на снимке, особенно при повторных запусках:

Х Удалить Профилирование кучи						
Идентификатор	Время	Выделения (разл.)	Размер ку	/чи (разн.)		
Профилировани	е машин	ной кучи включено	на 0,06 с,	предыдущие ра	спределения	
⇒ 1	0,06 c	0 (неприменимо)	0,00 КБ (неприменимо)		

Появилась информация о начальном состоянии памяти. Далее пройдём отладчиком до вызова функции foo_good и также выполним её (шаг с заходом F11):



После прохода функции сделаем снимок программы. Внутри данной функции выделялось и высвобождалось 4 мегабайта памяти, но можно видеть, что утечки памяти нет:



Теперь сделаем то же самое после функции foo_bad() и, дойдя до конца программы, тоже сделаем снимок памяти (синяя стрелка на скриншоте снизу):

•	15 16 17 18 19 20 21 22 23	<pre>int main() { // make memory screenshot foo_good(); // make memory screenshot foo_bad(); // make memory screenshot return EXIT_SUCCESS; S1мспрошло</pre>		100 0 Сводка Событ О Сделать снии Удалить	тия Ист мок Я	и Использование ЦП		
			V	1дентификатор	Время	Выделения (разл.)	Размер кучи (разн.)
				Ірофилировани	е маши	нной кучи включено	на 0,00 с, предыдуц	цие распреде.
				1	0,00 c	0 (неприменимо)	0,00 КБ (непри	именимо)
				1	0,00 c 0,01 c	0 (неприменимо) 0 (+0)	0,00 КБ (непри 0,00 КБ	1менимо) (+0,00 КБ)
				1	0,00 c 0,01 c 0,01 c	0 (неприменимо) 0 (+0) 1 (+1 ↑)	0,00 КБ (непри 0,00 КБ 4 096,04 КБ (+4 ^96)	именимо) (+0,00 КБ) ,04 КБ 🕇)
			_	1 2 3 • 4	0,00 c 0,01 c 0,01 c 0,01 c	0 (неприменимо) 0 (+0) 1 (+1 ↑) 1 (+0)	0,00 КБ (непри 0,00 КБ 4 096,04 КБ (+4 096, 4 096,04 К <u>с</u>	именимо) (+0,00 КБ) ,04 КБ 个) (+0,00 КБ)

Красная стрелка на скриншоте сверху указывает на состояние памяти после вызова функции foo_bad(), а зелёная, на состояние при завершении программы. Можно сделать вывод, что при выполнении программы выделяется лишние 4 мб памяти, причём в функции foo_bad().

Если нужно узнать, где именно в программе произошло выделение данной памяти. Для этого необходимо нажать левой кнопкой мыши на размер памяти (синяя стрелка на скриншоте выше):

После чего откроется окно с состоянием кучи в выбранный момент (моментальный снимок):

			100			•		100
Тип объекта	Счетчик	Размер (байт) 🔻	Сводка Событ	ия Ис	пользование	памяти	Использование	цп
int[]	1	4 194 304	🖸 Сделать сни	иок 🖌	Опросмотр	еть кучу		
			🗙 Удалить				🛱 Профилир	ование кучи
			Идентификатор	Время	Выделения	(разл.)	Размер кучи (р	азн.)
			Идентификатор	Время 0,01 с	Выделения	(разл.) іменимо)	Размер кучи (р 49,30 КБ (не	азн.) еприменимо
			Идентификатор 1 2	Время 0,01 с 0,01 с	Выделения 160 (непри 160	(разл.) іменимо) (+0)	Размер кучи (р 49,30 КБ (не 49,30 КБ	азн.) еприменимо (+0,00 К!
			Идентификатор 1 2 3	Время 0,01 с 0,01 с 0,01 с	Выделения 160 (непри 160 161	(разл.) іменимо) (+0) (+1 ↑)	Размер кучи (р 49,30 КБ (не 49,30 КБ 4 145,34 КБ (+4	азн.) еприменимо (+0,00 КЕ 096,04 КБ 🕇
			Идентификатор 1 2 3 — 4	Время 0,01 с 0,01 с 0,01 с 0,01 с	Выделения 160 (непри 160 161 161	(разл.) іменимо) (+0) (+1 1) (+0)	Размер кучи (р 49,30 КБ (не 49,30 КБ 4 145,34 КБ (+4 4 145,34 КБ	азн.) еприменимо (+0,00 КЕ 096,04 КБ 🕇 (+0,00 КЕ

Можно заметить, что создана запись об указателе на выделенную память. Далее можно перейти к самуму экземпляру, для этого выберите нужную строку и нажмите левой кнопкой мыши на значок, как указано стрелкой на скриншоте ниже:

Режим просмотра:	Предс	гавление типов		~	▲ Память пров 100	цесса	
Тип объекта		Счетчик	Размер (байт)	•	Сводка Собя	лтия Ис	пользование
int[]	9	1		4 194	🖸 Сделать сн	имок 🖌	О Просмотре
					🗙 Удалить		
					Идентификато	время	Выделения
						1 0,01 c	160 (непри

Теперь, нажав на нужный экземпляр, на стеке вызовов можно найти место в программе, где была выделена память (синяя стрелка на скриншоте ниже):

 Экземпляры int[] Моментальный снимок недоступны. 	не отражает текущее состо	яние программы. Значения	Ф ⊕ ⊕ € Q Сеанс диагностики: 14мс 14мс 4 События 4
Экземпляр	Размер (байт)	Возраст (мс) 🔻	
<0xCF0040>	4 194 304	3056,907000	Память процесса
			Сводка События
Стек вызовов выделений			Сделать снимок
Имя			🔎 Просмотреть куч
profiler.exe!foo_b	ad() — строка б		🗙 Удалить
profiler.exe!main() — строка 22	Фреймы стека	Идентификатор Вре
profiler.exelinvok	e_main() — строка 78		1 0,0

Дважды нажав на строку в стеке вызовов MS VS откроет окно с кодом программы и выделит оператор выделения данной памяти (синяя стрелка на скриншоте ниже):



Для более детального профилирования необходимо делать шаг с заходом в функцию и после каждой инструкции вызывать снимок состояния кучи.