очистки. Уровень развития биоценоза оценивается по преобладающим группам простейших, а деструкционный потенциал – по основным показателям функционирования аэротенков, таким как нагрузка на ил и степень очистки по биохимическому потреблению кислорода (БПК).

Показателями высокого уровня развития биоценоза, значит, и высокого качества очистки сточных вод, являются хищники: черви, сосущие инфузории, хищные коловратки, грибы, тихоходки. В таких условиях, когда высока степень удаления органических загрязнений, интенсивно протекает нитрификация.

Если на данный момент степень очистки сточных вод в аэротенке удовлетворяет предъявляемым требованиям, и при этом активный ил хорошо оседает во вторичных отстойниках, существующий биоценоз следует признать характерным для данных очистных сооружений.

Важным аспектом является своевременное выявление изменений в составе индикаторных групп организмов, которые могут указывать на снижение качества очистки. При ухудшении состояния биоценоза необходимо оперативно устранять причины таких изменений.

В условиях нехватки квалифицированных специалистов автоматизация анализа состояния биоценоза становится необходимой. Разрабатываемый сервис существенно упростит и ускорит работу, обеспечивая обработку данных в автоматическом режиме.

Сервис позволит не только улучшить контроль и управление процессами на конкретных очистных сооружениях, но и создать единую базу данных, охватывающую всю территорию Беларуси. Это позволит отслеживать общую ситуацию по очистке сточных вод, анализировать тенденции и оперативно реагировать на возникающие проблемы, что в итоге повысит эффективность работы всех очистных сооружений страны.

УДК 004.432.2

А.С. Наркевич, ст. преп. (БГТУ, г. Минск)

ПОИСК И ОБРАБОТКА ОШИБОК В C++

Существует два основных типа ошибок в программном коде: ошибки пользователя и ошибки программиста. Ошибка пользователя возникает, когда пользователь, например, вводит неверные данные, пытается открыть несуществующий файл и т.п. Ошибка программиста – это результат ошибки в коде.

Отладка – это способ диагностики дефектов в программе с целью

исправления ошибок. Поиск дефекта и его понимание обычно составляют 90% работы [1].

Для лучшего понимания кода и умения отслеживать ошибки в нём нужно, прежде всего, научиться читать и выполнять команды ассемблера в отладчике, отображать и анализировать значения переменных в окнах памяти или через регистры, просматривать стек вызовов (механизм, который отслеживает порядок вызова функций/методов в программе).

Ошибки при работе с памятью относятся к числу самых распространённых. Из них наиболее трудными для обнаружения являются утечки памяти и нарушение ее целостности, которые возникают, когда память, выделенная программой, но никогда не освобождается.

Отладчик Visual C++ и стандартная библиотека времени выполнения C (CRT) предоставляют набор инструментов для обнаружения и локализации утечек памяти. Чтобы активизировать эти функции, нужно включить в программу на C++ следующие директивы в указанном порядке:

```
#define _CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtdbg.h>
```

Деректива #define сопоставляет базовую версию функций кучи CRT с соответствующей отладочной версией. Без неё отчёт об утечках памяти будет менее подробным.

Включение файла crtdbg.h, сопоставляет вызовы функций malloc и free с их отладочными версиями _malloc_dbg и _free_dbg, которые отслеживают все операции по распределению и освобождению памяти. Перенаправление происходит только в отладочной версии программы (макрос _DEBUG должен быть определён). В окончательной версии будут использоваться обычные функции malloc и free.

Получить информацию об утечках памяти, можно вызвав функцию _CrtDumpMemoryLeaks().

Когда программа выполняется под управлением отладчика, _CrtDumpMemoryLeaks отображает информацию об утечках памяти на вкладке Debug окна Output.

Пример кода на C++:

```
#define _CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtdbg.h>
```

```
void main(){
char* pointer = nullptr;           // Создаем указатель
for (int i = 0; i < 5; ++i) {
     // На каждой итерации цикла в указатель помещаем новый
адрес
     pointer = new char[10];    // Создаем новый массив
}
delete[] pointer;         // Удаляем из памяти последний
массив

_CrtDumpMemoryLeaks();

}
```

Отчёт об утечках памяти:

```
Поток 0x3f8 завершился с кодом 0 (0x0).
Detected memory leaks!
Dumping objects ->
{84} normal block at 0x00F14BE8, 10 bytes long.
 Data: <          > CD CD CD CD CD CD CD CD CD CD
{83} normal block at 0x00F14BB0, 10 bytes long.
 Data: <          > CD CD CD CD CD CD CD CD CD CD
{82} normal block at 0x00F14B60, 10 bytes long.
 Data: <          > CD CD CD CD CD CD CD CD CD CD
{81} normal block at 0x00F149F0, 10 bytes long.
 Data: <          > CD CD CD CD CD CD CD CD CD CD
Object dump complete.
```

Подробнее использование инструментов описано в [2].

Иногда ошибки появляются только в релизных сборках, но они исчезают в режиме отладки. Эти ошибки бывают вызваны неинициализированными переменными, потому что переменные и динамически выделяемые блоки памяти часто устанавливаются в ноль в режиме отладки, но остаются «с мусором» в сборке без отладки.

Ошибки пользователя лучше всего обрабатывать, отображая некоторую полезную информацию для пользователя, а затем позволяя ему продолжать работать (с гибкой политикой «информируй и продолжай»).

Обнаружив ошибки программиста, лучше остановить выполнение программы и предоставить подробную информацию, чтобы программист мог быстро выявить и устранить проблему.

Распространённым подходом к обработке ошибок является возвращение некоторого кода ошибки из функции, в которой обнаружена проблема. Это может быть логическое значение, указывающее на успех

или неудачу, или значение, выходящее за пределы допустимого диапазона обычно возвращаемых результатов.

Коды возврата ошибок – это простой и надежный способ реагирования на ошибки.

Механизм обработки исключений в C++ позволяет функции, которая обнаружила проблему, сообщать об ошибке остальной части кода, ничего не зная о том, какая функция может обработать ошибку.

После обнаружения ошибки стек вызовов автоматически раскручивается в поисках вызывающей функции, чей вызов обернут в блок try-catch. Если найден блок try-catch, объект исключения сопоставляется со всеми возможными случаями catch и, если найдено совпадение, выполняется соответствующий блок кода catch. При этом деструкторы любых автоматических переменных вызываются по мере необходимости в процессе раскручивания стека.

Однако обработка исключений добавляет некоторые накладные расходы в программу.

Получение ресурса есть инициализация (RAII), это техника программирования C++, которая связывает жизненный цикл ресурса, который должен быть получен перед его использованием. Таким ресурсом может быть выделение памяти в куче, открытие потока выполнения, открытие сокета, открытие файла, выделение дискового пространства, подключение к базе данных и т.д. [3].

Использование RAII значительно упрощает управление ресурсами, уменьшает общий размер кода и помогает обеспечить корректность программы. Поэтому принцип RAII рекомендуется как безопасный метод управления ресурсами.

ЛИТЕРАТУРА

1. Макконнелл С. Совершенный код. Мастер-класс / Пер. с англ. – М. : Издательство «Русская редакция», 2010. – 896 с.

2. Найдите утечки памяти с помощью библиотеки CRT [Электронный ресурс] – URL: https://learn-microsoft-com.translate.goog/en-us/cpp/c-runtime-library/find-memory-leaks-using-the-crt-library?view=msvc-170&_x_tr_sl=en&_x_tr_tl=ru&_x_tr_hl=ru&_x_tr_pto=rq (дата обращения: 10.01.2025г.).

3. Resource acquisition is initialization [Электронный ресурс] – URL: https://en.wikipedia.org/wiki/Resource_acquisition_is_initialization (дата обращения: 10.01.2025г.).