УВАЖАЕМЫЕ СТУДЕНТЫ! Изучите приведенную лекцию, законспектируйте основные сведения о способах оптимизации программы.

Ответы на вопросы, фотоотчет, предоставить преподавателю на e-mail: r.bigangel@gmail.com до 06.03.2023.

При возникновении вопросов по приведенному материалу обращаться по следующему номеру телефона: (072)111-37-59, (Viber, WhatsApp), vk.com: https://vk.com/daykini

ВНИМАНИЕ!!! При отправке работы, не забывайте указывать ФИО студента, наименование дисциплины, дата проведения занятия (по расписанию).

Лекция № 37

Тема: «Оптимизация программы»

Цель: Изучить основные этапы оптимизации программы.

Основной задачей программирования является создание **правильных**, а не **эффективных** программ. Эффективная программа не нужна, если она не обеспечивает правильных результатов.

Правильную, но не эффективную программу можно оптимизировать и сделать эффективной.

Для больших программ на стадии проектирования определяются требуемые параметры, включающие время выполнения и емкость памяти. Оптимизация программы – это процесс построения по исходной программе эквивалентной программы, обладающей лучшими характеристиками времени работы и/или объема занимаемой ОП.

Существует 3 типа программ:

1. Часто используемые программы (ОС, компиляторы, постпроцессоры). Для них эффективность – первостепенная задача.

2. Производственные программы. Эффективность существенна.

3. Программы, написанные не программистами. Эффективность нужна если есть ограничения по памяти или по времени.

Программу делают более эффективной лишь в особых случаях:

1. Программа не помещается в памяти.

2. Программа слишком долго выполняется.

3. Программа должна быть включена в библиотеку и часто использоваться.

Шаги оптимизации:

1. Оптимизирующий компилятор.

2. Определить критические области, подлежащие оптимизации. Применить локальную оптимизацию в критических областях (наиболее часто используемых частях программы).

Компилятор – это программирующая программа, предназначенная для перевода (трансляции) описания алгоритмов с одного формального языка на другой. Первый из этих языков называется входным, второй - выходным или объектным. Наиболее распространены трансляторы с языков, процедурно-ориентированных в языки машинные и языки машинно-ориентированные.

Транслятор является одним из основных средств автоматизации программирования.

Обычно транслятор состоит из ряда блоков, выполняющих независимые функции:

1. Синтаксический анализ программ.

2. Анализ описаний данных и преобразование их в удобную для хранения и дальнейшего использования форму (в т.ч. вычисление констант).

3. Распределение памяти для используемых в программе объектов.

4. Установление соответствия между конструкциями входного языка и эквивалентными им конструкциями выходного языка.

5. Печать в отредактированном виде текста исходной программы.

6. Сообщение об обнаруженных в процесс трансляции ошибках в исходной программе.

7. Эквивалентные преобразования на уровне входного языка с целью оптимизации программы.

Для одного и того же входного языка целесообразно иметь несколько трансляторов, один из которых позволяет осуществить быструю трансляцию, выдавая менее эффективные программы. Такой транслятор можно использовать для отладки программ. Компиляторы, создающие эффективную объектную программу, обычно бывают большими и работают медленно.

Оптимизирующий транслятор, применяя различные методы оптимизации может получить более качественную объектную программу, используемую для многократных расчетов.

Обычно большая часть времени расходуется на выполнение очень небольшой части программы, но наиболее часто используемой, называемой, критической областью.

Как правило, только критическая область программы оптимизируется программистом вручную. Для обнаружения этих критических областей используются специальные средства, названные профилировщиками.

Применение процедур: подпрограмм и макросов.

Подпрограммы изначально появились как средство оптимизации программ по объёму занимаемой памяти — они позволили не повторять в программе идентичные блоки кода, а описывать их однократно и вызывать по мере необходимости. К настоящему времени данная функция подпрограмм стала вспомогательной, главное их назначение — структуризация программы с целью удобства её понимания и сопровождения.

Выделение набора действий в подпрограмму и вызов её по мере необходимости позволяет логически выделить целостную подзадачу, имеющую типовое решение. Такое действие имеет ещё одно (помимо экономии памяти) преимущество перед повторением однотипных действий: любое изменение (исправление ошибки, оптимизация, расширение функциональности), сделанное в подпрограмме, автоматически отражается

на всех её вызовах, в то время как при дублировании каждое изменение необходимо вносить в каждое вхождение изменяемого кода.

Даже в тех случаях, когда в подпрограмму выделяется однократно производимый набор действий, это оправдано, так как позволяет сократить размеры целостных блоков кода, составляющих программу, то есть сделать программу более понятной и обозримой.

Оверлейность программы. Под оверлейностью понимают возможность перенесения подпрограмм во время работы программы в быстродействующую память из некоторого другого типа памяти таким образом, что несколько подпрограмм в различное время занимают одну и ту же область памяти.

Недостатки:

- выигрыш в памяти за счет времени;
- дополнительные усилия программиста;
- сохранение промежуточных результатов.

Возможность оверлейности реализуется также при использовании виртуальной памяти.

Если машина имеет виртуальную память, то ОС автоматически делит программу на части фиксированной длины, называемые страницами. Кроме того, ОС, в случае необходимости, пересылает страницы в ОП. Таким образом проблема организации оверлейности перестает быть обязанностью программиста и возлагается на машину.