**Лабораторная работа 2.**

*Необходимые знания:*

* *Выделение и освобождение памяти в куче Си.*
* *В чем разница между стеком и кучей (прямо в задании не потребуется, но я спрошу).*
* *Использование аргументов командной строки*

**Стек**

Стек — это область оперативной памяти, которая создаётся для каждого потока. Он работает в порядке LIFO (Last In, First Out), то есть последний добавленный в стек кусок памяти будет первым в очереди на вывод из стека. Каждый раз, когда функция объявляет новую переменную, она добавляется в стек, а когда эта переменная пропадает из области видимости (например, когда функция заканчивается), она автоматически удаляется из стека. Когда стековая переменная освобождается, эта область памяти становится доступной для других стековых переменных.

Из-за такой природы стека управление памятью оказывается весьма логичным и простым для выполнения на ЦП; это приводит к высокой скорости, в особенности потому, что время цикла обновления байта стека очень мало, т.е. этот байт скорее всего привязан к кэшу процессора. Тем не менее, у такой строгой формы управления есть и недостатки. Размер стека — это фиксированная величина, и превышение лимита выделенной на стеке памяти приведёт к переполнению стека. Размер задаётся при создании потока, и у каждой переменной есть максимальный размер, зависящий от типа данных. Это позволяет ограничивать размер некоторых переменных (например, целочисленных), и вынуждает заранее объявлять размер более сложных типов данных (например, массивов), поскольку стек не позволит им изменить его. Кроме того, переменные, расположенные на стеке, всегда являются локальными.

В итоге стек позволяет управлять памятью наиболее эффективным образом — но если вам нужно использовать динамические структуры данных или глобальные переменные, то стоит обратить внимание на кучу.

**Куча**

Куча — это хранилище памяти, также расположенное в ОЗУ, которое допускает динамическое выделение памяти и не работает по принципу стека: это просто склад для ваших переменных. Когда вы выделяете в куче участок памяти для хранения переменной, к ней можно обратиться не только в потоке, но и во всем приложении. Именно так определяются глобальные переменные. По завершении приложения все выделенные участки памяти освобождаются. Размер кучи задаётся при запуске приложения, но, в отличие от стека, он ограничен лишь физически, и это позволяет создавать динамические переменные.

Вы взаимодействуете с кучей посредством ссылок, обычно называемых указателями — это переменные, чьи значения являются адресами других переменных. Создавая указатель, вы указываете на местоположение памяти в куче, что задаёт начальное значение переменной и говорит программе, где получить доступ к этому значению. Из-за динамической природы кучи ЦП не принимает участия в контроле над ней; в языках без сборщика мусора (C, C++) разработчику нужно вручную освобождать участки памяти, которые больше не нужны. Если этого не делать, могут возникнуть утечки и фрагментация памяти, что существенно замедлит работу кучи.

В сравнении со стеком, куча работает медленнее, поскольку переменные разбросаны по памяти, а не сидят на верхушке стека. Некорректное управление памятью в куче приводит к замедлению её работы; тем не менее, это не уменьшает её важности — если вам нужно работать с динамическими или глобальными переменными, пользуйтесь кучей. Вкратце, стек — это очень быстрое хранилище памяти, работающее по принципу LIFO и управляемое процессором. Но эти преимущества приводят к ограниченному размеру стека и специальному способу получения значений. Для того, чтобы избежать этих ограничений, можно пользоваться кучей — она позволяет создавать динамические и глобальные переменные — но управлять памятью должен либо сборщик мусора, либо сам программист, да и работает куча медленнее.

**Выделение динамической памяти в C**

void \* malloc( size\_t sizemem );

*Описание*

Функция malloc выделяет блок памяти, размером sizemem байт, и возвращает указатель на начало блока.

Содержание выделенного блока памяти не инициализируется, оно остается с неопределенными значениями.

*Параметры:*

sizemem

Размер выделяемого блока памяти в байтах.

*Возвращаемое значение*

Указатель на выделенный блок памяти. Тип данных на который ссылается указатель всегда void\*, поэтому это тип данных может быть приведен к желаемому типу данных.

Если функции не удалось выделить требуемый блок памяти, возвращается нулевой указатель.

void \* calloc( size\_t number, size\_t size );

*Описание*

Функция calloc выделяет блок памяти для массива размером — num элементов, каждый из которых занимает size байт, и инициализирует все свои биты в нулями.

В результате выделяется блок памяти размером number \* size байт, причём весь блок заполнен нулями.

*Параметры:*

number

Количество элементов массива, под который выделяется память.

size

Размер одного элемента в байтах.

*Возвращаемое значение*

Указатель на выделенный блок памяти. Тип данных на который ссылается указатель всегда void\*, поэтому это тип данных может быть приведен к желаемому типом данных.

Если функции не удалось выделить требуемый блок памяти, возвращается нулевой указатель.

void \* realloc( void \* ptrmem, size\_t size );

*Описание*

Функция realloc выполняет перераспределение блоков памяти.

Размер блока памяти, на который ссылается параметр ptrmem изменяется на size байтов. Блок памяти может уменьшаться или увеличиваться в размере.

Эта функция может перемещать блок памяти на новое место, в этом случае функция возвращает указатель на новое место в памяти. Содержание блока памяти сохраняется даже если новый блок имеет меньший размер, чем старый. Отбрасываются только те данные, которые не вместились в новый блок. Если новое значение size больше старого, то содержимое вновь выделенной памяти будет неопределенным.

В случае, если ptrmem равен NULL, функция ведет себя именно так, как функция malloc, т. е. выделяет память и возвращает указатель на этот участок памяти.

В случае, если size равен 0, ранее выделенная память будет освобождена, как если бы была вызвана функция free, и возвращается нулевой указатель.

*Параметры:*

ptrmem

Указатель на блок ранее выделенной памяти функциями malloc, calloc или realloc для перемещения в новое место. Если этот параметр — NULL, просто выделяется новый блок, и функция возвращает на него указатель.

size

Новый размер, в байтах, выделяемого блока памяти. Если size равно 0, ранее выделенная память освобождается и функция возвращает нулевой указатель, ptrmem устанавливается в 0.

*Возвращаемое значение*

Указатель на перераспределенный блок памяти, который может быть либо таким же, как аргумент ptrmem или ссылаться на новое место.

Тип данных возвращаемого значения всегда void\*, который может быть приведен к любому другому.

Если функции не удалось выделить требуемый блок памяти, возвращается нулевой указатель, и блок памяти, на который указывает аргумент ptr остается неизменным.

void free( void \* ptrmem );

*Описание*

Функция free освобождает место в памяти. Блок памяти, ранее выделенный с помощью вызова malloc, calloc или realloc освобождается. То есть освобожденная память может дальше использоваться программами или ОС.

Обратите внимание, что эта функция оставляет значение ptr неизменным, следовательно, он по-прежнему указывает на тот же блок памяти, а не на нулевой указатель.

*Параметры:*

ptrmem

Указатель на блок памяти, ранее выделенный функциями malloc, calloc или realloc, которую необходимо высвободить. Если в качестве аргумента передается нулевой указатель, никаких действий не происходит.

*Возвращаемое значение*

Функция не имеет возвращаемое значение.

*Необходимые знания*

* Основы работы компилятора: препроцессор, компилятор, линковщик. Их роли и порядок работы.
* Что такое статическая и динамическая линковка. В чем разница?
* Опции компилятора: -I, -L, -l, -shared, -o, -с, -fPIC
* Утилита ar.
* Переменная окружения LD\_LIBRARY\_PATH

**Препроцессор** — это компьютерная программа, принимающая данные на входе и выдающая данные, предназначенные для входа другой программы (например, компилятора).

**Компиля́тор** – это специальная программа, которая переводит текст программы, написанный на языке программирования, в набор машинных кодов.

**Компоновщик** (также редактор связей, от англ. link editor, linker) — инструментальная программа, которая производит компоновку («линковку»): принимает на вход один или несколько объектных модулей и собирает по ним исполнимый модуль.

**Объектный файл** – файл, содержащий скомпилированный машинный код и дополнительные данные (зависимости, как загружать и запускать и т.д.)

**Сборка:**

1. **Препроцессинг**

**C/C++ с директивами препроцессора -> Чистый код**

1. **Компиляция**

**С/C++ -> Ассемблер -> Машинный код**

**Один исходный файл -> Один объектный файл**

1. **Компоновка**

**Объектные файлы -> Исполняемый файл**

**Статическая библиотека** это фактически архив объектных файлов, который используется в процессе статической линковки. В результате статической линковки из многих объектных файлов получается один исполняемый, запускается статическая линковка в момент создания исполняемого файла.

**Динамическая библиотека** это фактически исполняемый файл (т.е. DLL и EXE в Windows имеют одинаковый формат). Динамическая линковка запускается в момент создания процесса (когда вы запускаете исполняемый файл на выполнение), линкуются между собой несколько исполняемых файлов каждый раз, когда создается новый процесс. Так же возможна динамическая линковка уже после запуска, т.е. новая библиотека может быть подгружена в адресное пространство ужа работающего процесса.

Сделать из статической библиотеки динамическую в принципе можно - необходимо ее (статически) слинковать в динамическую библиотеку, при этом будет создана динамическая библиотека и статическая стаб-библиотека, которую можно использовать в проекте вместо статической библиотеки, чтобы вызывать функции из динамической библиотеки.

**Опции компиляции**

Среди множества опций компиляции и компоновки наиболее часто употребляются следующие:

-c Эта опция означает, что необходима только компиляция. Из исходных файлов программы создаются объектные файлы в виде name.o. Компоновка не производится.

-o file-name Использовать file-name в качестве имени для создаваемого файла.

-I Используется для добавления ваших собственных каталогов для поиска заголовочных файлов в процессе сборки

-L Передается компоновщику. Используется для добавления ваших собственных каталогов для поиска библиотек в процессе сборки.

-l Передается компоновщику. Используется для добавления ваших собственных библиотек для поиска в процессе сборки.

-shared создает общий объектный файл для общей библиотеки.

-fPIC

генерирует независимый от позиции код (PIC) для разделяемых библиотек.

**ar** — стандартная утилита Unix, архиватор, не использующий сжатия данных.

ar [-]{dmpqrtx}[abcilosuvV] [имя\_компонента\_архива] архив-файлы…

Пример создания архива

$ ar r texts.ar \*.txt

ar может не только создавать/извлекать файлы из архива, но и модифицировать их.

Содержимое оригинального файла, права доступа, время, владелец и группа сохраняются в архиве и могут быть переопределены при извлечени

d Удаляет модули из архива

m Перемещает файлы (перемещаются в конец архива)

p Выводит заданные компоненты архива на стандартный вывод

q Добавляет файлы в конец архива, без проверки на замещение

r Вставляет файлы в архив (с замещением)

t Показывает таблицу, в которой хранится содержание архива, или файлы, что были указаны в качестве параметра и присутствуют в архиве

x Извлекает компоненты (данные как файлы) из архива

a Добавить новые файлы после одного из существующих в архиве компонента

b Добавить новые файлы перед одним из существующих в архиве компонентов

c Создать архив

f Урезает имена в архиве для совместимости с другими операционными системами

i Вставить новые файлы перед одним из существующих в архиве компонентов

o Восстанавливает оригинальную дату компонентов архива, когда они извлекаются

s Записывает индекс объектного файла в архив или, если он существует, обновляет его, даже если нет других изменений в архиве

u При замещении файлов вставить только те из них, что отличаются от уже имеющихся в архиве

v Включает режим выдачи подробностей при выполнении операции

V Показывает версию ar

**LD\_LIBRARY\_PATH** - специальной переменная среды, в которой перечисляются все каталоги содержащие пользовательские динамические библиотеки.

echo $LD\_LIBRARY\_PATH