https://docs.google.com/document/d/1zadupW0wYqSp80CHE2yQOl4iQKtYJ9c-F6Uq7lF1JF0/edit

0. Comparisons

Enumerated Types:
Arithmetic types used to define variables that can only assign certain discrete int values
throughout the program
It is a way to create a type that is just a list of keywords. Compiler treats list of keywords
as integers
Easy trick to print out string representation is to just use an array with indexes
corresponding to the level of the member in the enum
Enumerated types, on the other hand, made the code more self-documenting. Depending on the language, the compiler could automatically assign default values to the enumerators thereby hiding unnecessary detail from the programmer. These values may not even be visible to the programmer (see information hiding).

> Inline replaces a call to a function with the body of the function, however, inline is just a *request* to the compiler that could be ignored (you could still pass some flags to the compiler to force inline or use *always_inline* attribute with gcc).
>
> A macro on the other hand, is expanded by the *preprocessor* before compilation, so it's just like text substitution, also macros are not type checked, inline functions are.

structs vs. macros Macros are not type safe. Before it compiles and checks types. ^if p is a pointer or a structure, !=0 doesn't mean anything. Compare 0.0 and char, will
get into trouble. Use macros sparingly.

Shell Commands: cd, ls ,less, more, pwd
Less is a program similar to *more* (1), but which allows backward movement in the file as well as forward movement.
More is a filter for paging through text one screenful at a time.
The **pipe**(8) daemon updates queue files and marks recipients as finished, or it informs the queue manager that delivery should be tried again at a later time.
pwd is Print working directory
Use “cd -” to toggle between the last two directories
chmod command is used to change the permissions for a file or directory.
mkdir

What is the difference between struct and union?
Union -> is only over one piece of data. Structs bind multiple pieces of data under one name. Unions allow multiple ways to access one chunk of data. So if you write
an int into a union, and you have char. It will give you the int as a char. Struct every variable is a different piece of data.
Not frequently used.

What is the difference between file descriptors and file handlers?

A file descriptor is a data structure that is used to access a file.

A file handler indicates a position to reference within the file. (ex. read 10 bytes, read another 10 bytes. this is the file handler, the position of the

file handler in the parent process will be in the same position)

Do not use file descriptors or handlers directly after you fork.

In simple words, when you open a file, the operating system creates an entry to represent that file and store the information about that opened file. So if there are 100 files opened in your OS then there will be 100 entries in OS (somewhere in kernel). These entries are represented by integers like (...100, 101, 102....). This entry number is the file descriptor. So it is just an integer number that uniquely represents an opened file in operating system. If your process opens 10 files then your Process table will have 10 entries for file descriptors.

1. Exceptions to the Rules

Exec is another strange function, because it is called. Where does exec return to? Exec does not return.

Functions that do not return values have a return type of void.

fork: called once, returns twice (parent and child)

exec: called once, normally does not return (replaces the call to exec in the child with new code)

signal handlers: is not called, but does return (because it is invoked by the OS on your behalf)

Signal handler functions -- invoked, but not called. Returns to next instruction without call instruction.

2. Process Life Cycle

1. **FROM CODE TO AN EXECUTABLE PROGRAM:** for code written in c language, you have to compile it to create an executable program (gcc -Wall helloWorld.c -o world)
2. **FROM AN EXECUTABLE PROGRAM TO A PROCESS:** an executable program is a passive entity that does nothing until it is run. when it is run, a new entity is created which is a process. (command pstree displays hierarchy of active processes in linux system, ps command on linux is one of the most basic commands for viewing the processes running on the system. It provides a snapshot of the current processes along with detailed information like user id, cpu usage, memory usage, command name)
3. **DIFFERENT STATES OF A LINUX PROCESS:** dynamic view of a process (top command gives you real time view of linux system in terms of processes and can also see state) Four states:

a. RUNNING -> either is actually running/executing or waiting in the scheduler's queue to get executed

b. WAITING OR SLEEPING -> process is waiting for some resource specific operation to complete
c. STOPPED -> it has received the signal to stop (usually program is b/c program is being debugged)
d. ZOMBIE -> process has finished execution but is waiting for parent to retrieve its exit status

What does fork() do?
It generates a duplicate of current process. The only difference between the proceses is that in the forked process that address space, the process id, and the parent
process id are different. These are differeneces that are present immediately after the fork.

Significant similarities (Immediately after Fork):
(1) Running the exact same code
(2) Same variables/data
(3) file descriptors/file handlers
(4) signal disposition

Fork is called once, but returns twice for parent and for the child.

How do you know if you are the parent or the child process?
Return values from fork are different:
child: 0
parent: child's PID

If you are waiting for fork to return and it returns 0, you are the child process. If you get something other than 0 than you are the parent process.

What is pid 0?
The very first thing that you run. It has no parent. It is the scheduler. It knows how to run on pieces of code, it is done by the OS.

Pid 1 is the init process, which is the process that knows how to build processes. Init is the first fork.

If you are the child process (PID 0)
If you generate a process probably want to do something different. If you are child you should execute, loads some file.
On exec call, all vestiges of the parent are dumped.

Exec is another strange function, because it is called. Where does exec return to? Exec does not return.

fork: called once, returns twice (parent and child)
exec: called once, normally does not return (replaces the call to exec in the child with new code)
signal handlers: is not called, but does return (because it is invoked by the OS on your behalf)

What is a signal? It halts execution and indicates a special event the OS needs to address. It may be an error or it might not.
Resizing a window will result in a signal. (stops execution and redraws everything) SEGFAULT is a signal. Can set a timer that will
halt execution after a certain amount of time. KILL is a signal (halts the execution). Nothing is being run while the signal is being processed.

Signal Handler is code you write to be run in the event of some particular signal, or group of signals (similar to exceptions in java)
-> When signal handler is done running, you return to back to exactly where you were.

++++++++++++++++++
Exception Model -> jumps past the problem
++++++++++++++++++
try{
... code
}
catch(ERROR) -> when this happens you can not go back
{
...run this code
}
...continue here

+++++++++++++++
Signal Model -> returns you right to where the signal occurred .. resumes from where it left off (returns to the problem, goes back to instruction 3 here)
+++++++++++++++
instruction 0
instruction 1
instruction 2
instruction 3
SIGNAL! ... halt execution .. trap to OS ... invoke signal handler .. run signal handler to completion .. return ...
instruction 4

Trap means go run the operation system code that knows what to do next

What happens if in your signal handler, a signal occurs?
If it is the same signal, the OS will block that signal. You can instruct it to not do that (need a really good reason).

What's a zombie process?

When you kill the parent process before it can do the wait system call. Zombie process is a child process that is done computing,
but the parent process never called wait() ont it.

Who is the system-wide foster parent?
...if the parent process is killed, who takes over parent duties for the child process?
Init process, pid 1. Init takes over.

exec functions : execcl(), execcv(), execle(), execve(), execlp(), execvp().

The execv function executes the file named by filename as a new process image.

3. Danger!
https://www.eskimo.com/~scs/cclass/int/sx7.html
Malloc() & Free()
Pointer Arithmetic

Must initialize things before you use them, have to clean them up when you are done.

The memory allocation and deallocation functions could be written to keep track of what has been used and what has been freed. Typically, they aren't. If you free() a pointer, the pointed-to memory is assumed to have been allocated by*malloc()* or *calloc()* but not deallocated since then. *free()* calculates how big that chunk of memory was and updates the data structures in the memory "arena." Even if the memory has been freed already, *free()* will assume that it wasn't, and it will blindly update the arena. This action is much faster than it would have been if *free()* had checked to see whether the pointer was OK to deallocate.

What happens when you free a pointer twice?
Doing a free on an already freed memory will result in double free memory corruption.
The `free` function causes the space pointed to by its argument to be deallocated, that is, made available for further allocation. If the argument is a null pointer, no action occurs. Otherwise, if the argument does not match a pointer earlier returned by a memory management function, or if the space has been deallocated by a call to free or realloc, the behavior is *undefined.*
Double free occurs mostly when there are two pointers pointing to the same address.

Why shouldn't you return a pointer to a local variable? (?)
Local variables disappear as soon as the function call returns. Popped off the stack. It will point to nothing. That's a problem.
What happens when you never free anything?
Not freeing an object when you're done with it will mean the destructor will never get called, and any resources the class is responsible might not get cleaned up. Garbage will hang around.

First of all, of course, you must always ensure that the allocation functions malloc and realloc succeed. These functions return null pointers when they are unable to allocate the requested memory, so you must *always* check the return value to see that it is not a null pointer, before using it. (If the return value is a null pointer, you will generally print some kind of error message and abort at least the particular function that needed the allocated memory, or perhaps abort the entire program.)

After calling free, is p valid or invalid? C uses pass-by-value, so p's value hasn't changed. (The free function couldn't change it if it tried.) But p is most definitely now *invalid*; it no longer points to memory which the program can use. However, it does still point just where it used to, so if the program accidentally uses it, there will still seem to be data there, except that the data will be sitting in memory which may now have been allocated to ``someone else''! Therefore, if the variable p persists (that is, if it's something other than a local variable that's about to disappear when its function returns, or a pointer field within a structure which is all about to disappear)

A final issue concerns pointer aliases. If several pointers point into the same block of memory, and if that block of memory moves or disappears, *all* the old pointers become invalid.

#ifndef thing (if not defined, do not want to define a thing that is already defined)
#define header.h

```
  typedef enum neato = {THINGS} thing;
```

#endif

4. Total Recall
PA2 (necessary to keep …

Lecture Notes about PA3:
Project Stuff

Don't malloc inside of malloc

Linked List Implementation:

typedef struct stuff{
prev*
next*
size
void *data
}MemEntry;

You don't need to tell it type, you only need to tell

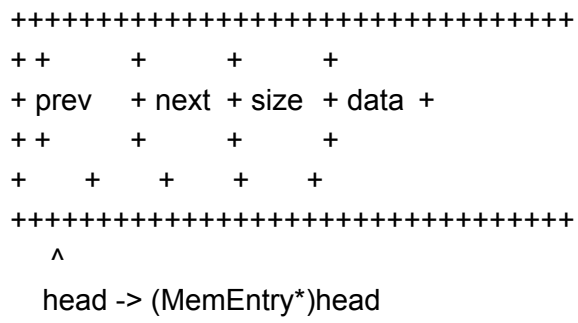
it the size of the thing you want to store. Data should be
void*

(int*)malloc(sizeof(MemEntry)) ->gives you a pointer.
what type of pointer?

You want to dynamically allocate size of MemEntry. What
does malloc return? It returns a pointer to a block of memory
of type void*

Before yous start using this, you should check if its null
so that you check if malloc has worked.

ptr + 5



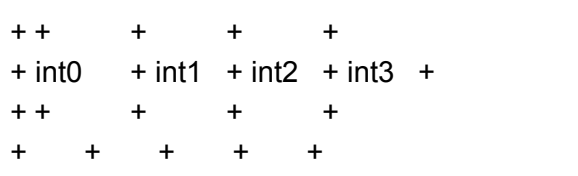
Take the head of your list and cast it.

Malloc, give me enough space to store this struct, here's a pointer
to the stored space

={1,4,6,7};

Puts each value in array in order. Can think of an array as a struct where you
don't give names to the value, you just know where they are.

If you say size of that, it gives you a pointer to the beginning of that space.



++++++++++++++++++++++++++++++++

^
|
(int*)head

How to get to end of struct, if you take pointer and do pointer + 1 and it will go to end of list and it will take you to the end of the struct

What does malloc do? You need a block of memeory of a certain size. It should give you a pointer that is of that size. So if you are malloc, you can say okay well I there is no previous or next block of memory and size is 5000. You only allocated size of memEntry which means you allocated a chunk of 5000 byte long chunk of data. You do not have 5000 to give the user some has already been used by mementry

User wants space for 10 integers. Malloc gives you backa void*. It gives you memory of size you want and gives you a pointer to the space you are allowed to modify. Return a pointer to bytes. Where will it point to and how will I build it?
k is size actually the size of the thing the user wanted? But user didn't ask for anything wants. It will be the size of the block that contains everything. Need to know how big the block is.
Add a free.

Give user a pointer to the space at the end of the size block. Do head + 1, and this will go the length of one mementry and then pass this pointer to the user and then update some internal stuff. You make a mementry struct every time.
Need some way to control the rest of my space

If int pointer, and add two, it will increase by two. points to second integer. ptr + 2.

If head is of type pointer to mementry, head+1 will give you the next mementry. All you do is give them a pointer. User uses the pointer, not malloc. RAM Chip is
5000 byte. That's all you've got. If they want 10,000 bytes, you return 0.

MemEntry blocks, track the blocks of memory. Whether they are free or allocated, you still ahve to track them. Free 0 means its been allocated. Free 1 means
it has not allocated.
declare a statically sized array at the begining.

5.

C pointer arithmetic

Given

```
char word[10];
char *cptr;
```

cptr = word;

Each row in following table gives equivalent forms

cptr	word	&word[0] //address
cptr + n	word + n	&word[n] //address
*cptr	*word	word[0] //value
*(cptr + n)	*(word + n)	word[n] //value

Be careful when you are computing addresses

Address calculations with pointers are dependent on the size of the data the pointers are pointing to

Examples:

```
int *i; …; i++;        /* i = i  + 4 */
char *c; …; c++;   /* c = c + 1 */
double *d; …; d++;            /* d = d + 8 */
```

Another example:

```
double x[10];  // double = 8 bytes
double *y = x; // y becomes pointer to beginning of array
*(y + 3) = 13;        /* x[3] = 13 */
```