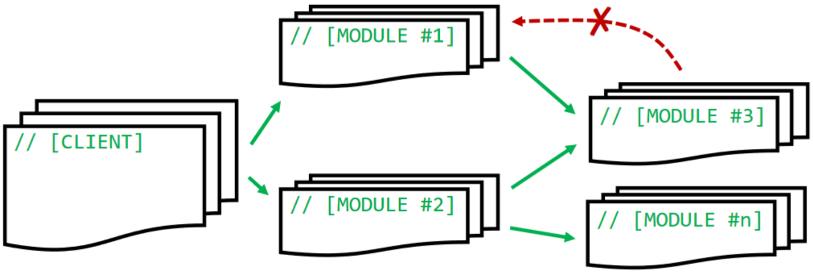
By Strafe <3

Entry Point	**Special function named main**. By calling main, **OS transfers control flow** to the program. After execution, the control flow r**eturns to OS.**
Return Type	Data type returned to caller
Identifier	Name by which the function, variable, structure is called. Generally, it must be unique
Parameter list	Where a function can receive data from the caller. Each parameter consist of the data type followed by identifier
Expression	Computer program statement that evaluates to some value. E.g. x + y
Static type	**All types must be known when the program compiles.** The type of an identifier **cannot be changed.**
Data	Stored in memory as a sequence of bits (1 and 0). Eight bits form one byte
Primitive Data Types (Primitives)	Integer, Characters, Boolean values, Floating point numbers
Order of precedence	**A collection rules that reflect conventions** about which **performs first in order to evaluate a given expression**
Side effect	Occurs **when the state of something changed. State refers to the value of some data** at a moment in time. E.g. I/O Mutation (pointers, variables)
Arguments	Values that are **declared within a function when the function is called.**
Parameters	Variables defined when the **function is declared**
Data scope	Region of code where it is accessible. Local (in the function) Global (outside the function)
Variables	To store mutable state information (values)
Control flow	Order of instructions the program executes - Function calls - conditions - iterations
Return statement	- **Stops the function returns to the line** of code that called it - For non void functions, r**equire an expression followed, expression determines the value of the function call**

	expression - Void function, not followed.
Uninitialized variable	Global, set to 0 Otherwise: arbitrary
Overflow	When you try to represent a value outside the range of possible values. Occurs in C because values are allocated in a fixed number of bytes.
Break	Control flow statement that indicates you want to exit from the middle of the loop. Terminates the innermost loop.
Continue	Control flow statement that indicates you want to skip over the rest of the statements in the code block and continues with the next iteration.
Mutation	A side effect, alter its value after it has been defined. When a variable's value is changed etc.
Aliasing	Multiple pointers point to the same data in memory
Dereferencing	"*" Accessing or manipulating data stored in a memory location pointed to by a pointer

—----MIDTERM END—-----

Modularization	Dividing the program into well-defined modules. separating implementation from interface and hiding information in the implementation
Advantages of Modularization	- Reusability - Able to construct larger programs more easily, buy or license third-party modules) - Maintainability - Much easier to test and debug a single module instead of a larger program using a testing suite - Abstraction - The client only neds an abstraction of how it works can write large programs without having to understand how every piece works. Can replace entire module without knowing.
Terminology	CLIENT **requires** the function that a MODULE **provides**

Module dependency graph	The module dependency graph should not have any cycles.  // [CLIENT] // [MODULE #1] // [MODULE #2] // [MODULE #3] // [MODULE #n]
Declaration	Simply introduces an identifier [NOT ALLOCATING MEMORY] E.g. Extern int math_sqrt(int n);
Definition	Gives content to an identifier (contains an identifier). Identifier can be declared multiple times ut only defined once. [ALLOCATING MEMORY]

Incomplete declaration vs complete devlaration	Struct posn; <- incomplete (opaque)
	
	Struct posn {
	Int x;
	Int y;
	}; <- complete (transparent)

Interface (Header File)	The .h File Only module declarations
Implementation (Source File)	The .c file module definitions
Information Hiding	- Increases security because it prevents clients from direct access to data stored within a module. May interact with interface only - Increases flexibility, because it allows for changing the underlying implementation of a module without affecting client (if interface remains unchanged)

Opaque Structure	Struct posn;
	
	Struct posn{
	Int x, y;
	}
	- Clients have no information about or access to structure fields.
	- Incomplete Declaration
	- Only pointers to an opaque structure can be defined

Transparent Structure	Put the complete declaration of struct in interface file.

Data Structure VS ADT	Data structure: know the data is structured, can access the data directly ADT: does not know how data is structured, can only access through interface
Stack operations	- Push - Top (peek) - Pop - empty?
Queue operations	- Enqueue - Dequeue - Front (peek) - Empty?
Sequence	- Length - Insert - At - Remove
Oversize array	Need to know the length in advance Wasteful if Maximum is excessively large Restrictive if the maximum is too small. Have to keep track of curr_len, max_len
Null Terminator	‘\0’. The end of a string is determined by the location of the null terminator.
String Literal	Strings that are not initialized as an array. For each string literal, a null terminated const char [] is created in the global read only section of the memory (global constants). Occurrence of the string literal is replaced with address of array.
String VS String Literal	String Literal: Content immutable, identifier reassignable, String: Content mutable, identifier not assignable E.g. Char *str_lit = “CS 136” Char *another_str_lit = “fortnite” Str_list = another_str_lit
Heap	Memory is allocated from the heap upon request. If too much memory has already been allocated, attempts to borrow additional memory fail
Heap Advantages	- Dynamic - Resizable - Scope: Memory persists until freed, func can allocate memory and still be valid upon returning - Safety: If memory runs out, can be detected and handled properly

Heap Out of Memory Error	Unsuccessful call to malloc returns NULL, good style to check for NULL instead of crashing.
Free	Once freed, reading from or writing to it is invalid, and may cause errors or unpredictable results. Freeing it again after freed may cause errors or unpredictable results Error: double-free

Dangling Pointer	A pointer to a freed allocation.
	
	Int *data = malloc (4 * sizeof(int));
	free(data);
	Data = NULL;
	free(data); // NO ERROR!
	
	Advisable to assign NULL to a dangling pointer.

Runtime Error	Freeing memory that was not returned by malloc. Error: Non malloced address
Memory Leak	Occurs when allocated heap memory is not freed before program's termination. - Suffer degraded performance - Eventually crash
Garbage Collector	Detects when memory is no longer in use and automatically frees memory and returns it to the heap. Disadv: Slow, affect performance
Documentation of Dynamic Mem	Allocating and deallocating memory has a side effect: it modifies the state of the heap. Gotta write: **Effects: allocates heap memory [caller must free]** Freeing also has side effect **Effects: data becomes invalid**
Realloc	Preserves the content from the old array and resizes it. Time: O(n) **Note:** pointed returned by realloc may be the original pointer depending on circumstances. Only the new returned pointer can be used. Also, if size is smaller than OG size, extra memory is discarded.
Doubling Strategy	Double the length of the array when the current array is full. Have to keep track of actual length and allocated length.
Linked List	Sequence of nodes, where each node contains some data and a link to the next node in the list. Last node does not link to another node. - Can grow and shrink at runtime - Easily add items to and remove from front and middle
Data Integrity	- Introduces new ways that the data can be corrupted - Advance testing methods can mitigate - Repackage it into a module, so only the interface can be

	interacted with
Void Pointers	“Generic” type, stores address of any type of data (except functions) CANNOT BE DEREFERENCED but can be assigned to any pointer type variable and hten be dereferenced. (e.g. malloc)

Assigning Void Pointers	Converting a void pointer into a pointer at any data type, break static typing -> stack bufferoverflow
	e.g.
	Int i = 0;
	Void *ptr = &i;
	Struct posn *posn = ptr;

Good Style	Name void pointer with correct type Void *int_ptr = &xxx;
Void pointers as params	Generic Function: (void (*) (void *)) void double_int(void *vptr)
Wrapper Strategy	Pointer to the back
Node Augmentation	Have additional augmentations to store more data in each node
Selecting Data Structure	- How frequently you add / remove - How frequently will you search - Access item at specific pos? - Preserve original sequence? - Can you duplicate?
Order of Data	E.g. “A” “B” “C” String in an ADT If “A” is first place, it is **sequenced** If its a guest list, it doesnt matter, so its **unsequenced or rearrangable**
Sequenced Data	- Dynamic Array - Linked List
Unsequenced Data	- Sorted Dynamic List - Sorted Linked List - BST - Self balancing BST
Array	Good for frequently accessing elements at specific position
Linked List	Good for adding and removing at start / front
Self Balancing BST	Good for unsequenced data to frequently search / add or remove

Sorted Array	Rarely add items, frequently search for elements in sorted order

Functions	
String handling <string.h>	Strcat, strcmp, strcpy, strlen, strncat, strncmp, strncpy
Assertion <assert.h>	assert
Limits <limits.h>	INT_MAX, INT_MIN
IO <stdio.h>	Scanf, NULL
Bool <stdbool.h>	True False
Memory <stdlib.h>	Free, malloc, realloc, bsearch, qsort, exit, NULL, EXIT_SUCCESS, abs
Generic	Memcpy