Compiling

The preprocessor expands macros and includes files ('-E'). In the compiling stage, the source code is parsed and turned into an intermediate representation ('-emit-llvm -S'). Machine-specific assembly code is generated ('-S'). Machine code is generated in an object file ('-c'). The linker combines object files and libraries to create an executable, checks all functions have machine code available, complains if not. 'clang -o hello hello.c'. '-Werror' makes all warnings errors. '-Wall' enables all warnings.

Fundamental of C

int main (int argc, char* argv[]). %d is a decimal integer. %s is a string. %c is a character. %f is a floating point number. %.f is a double-precision floating point number. char 1 byte, int 4 bytes, float 4 bytes, double 8 bytes. Strings have a null terminator, one extra byte: '\0' Use strcmp to compare strings. include <stdbool.h> for bools. a pair of curly braces is a block, and introduces a lexical scope. variables declared inside a block are local to that block. Lifetimes, automatic: ends at the end of the block, static: ends at the end of the program. allocated on the stack, or heap, at runtime, using dynamic memory allocation (malloc). Returining pointers to local variables is dangerous, as the local variable will go out of scope when the function returns and the pointer will point to invalid memory. Stack is last in first out. When a block enters, its variables are allocated on the stack, When it exits, its variables are deallocated. The stack is automatically managed by the compiler, the size of every variable is fixed at compile time. Fixed size arrays are allocated on the stack. Dynamic arrays are allocated on the heap, using malloc for allocation and free for deallocation. Struct is sequence of members, passed by value. Array, passed by reference. Challenges with manual memory management: double free errors, dangling pointers, memory leaks. 64-bit architecture: addresses are 64-bit. In practice, 48 bits are used. Dereferencing a pointer: Access the value stored at the memory address. Dereferencing a void pointer: Forbidden as the type of the pointer is unknown, the user must cast the pointer to the correct type before dereferencing it. Make a pointer (int* p = &x). int param[] and int *param are the same. float * const ptr: ptr is a constant pointer to a float. const float * ptr: ptr is a pointer to a constant float. const float * const ptr: ptr is a constant pointer to a constant float. ptr[i] and * (ptr + i) are the same. `ptr->member` is the same as (*ptr).member. Dereferencing a pointer is unsafe. Dereferencing a void pointer is forbidden. Heap is part of memory reserved for dynamic allocation. It is safe to call free (NULL). If we don't call free, we leak memory. Returning a pointer to a local variable is dangerous. Function pointers `(return type (*func) (arg_types))`.

Errors

Segmentation faults: accessing memory that is not allocated, or accessing memory that is protected. Causes for segmentation faults: Dangling pointers, dereferencing NULL, writing to read-only memory, buffer overflow, stack/heap overflow. Memory Sanitizer, Address Sanitizer, Leak Sanitizer. To check where an error occurs, use GDB, use -g flag when compiling.

Memory Management and Ownership

RAII: Resource Acquisition Is Initialisation. Use constructor to allocate memory, destructor to free it. For storing a value, use unique_ptr for unique ownership, shared_ptr for shared ownership. Ownership is the concept of managing the lifecycle of resources, particularly memory.

Concurrency

Concurrency is about dealing with lots of things at once. Concurrency is a programming paradigm. Parallelism is about doing lots of things at once. Parallelism is about making programs faster. Processes are instances of a program. Threads are instances of a process. Multiple threads can be executed simultaneously. Threads share the same address space. Processes have their own address space, the OS ensures this. Mutual exclusion is the mechanism that ensures that only one thread can access a resource at a time. Mutual exclusion is used to protect critical sections of code. Example showing the need for mutual exclusion: removal of elements from a linked list. Critical region is the part of the code that updates some shared state. Locks: before entering a critical region, acquire a lock. After leaving a critical region, release the lock. Deadlock: two threads are waiting for each other to release a lock. Bust Waiting: one thread is waiting for another thread to release a lock, wastes CPU cycles. use condition variables to wake up threads that are waiting for a condition to be true. Important thread coordination aspects: partitioning: what parts of the computation should be separatel evaluated, data sharing: what data to share between threads, synchronisation: ensuring threads can cooperate without interference. Semaphores: a semaphore is a variable that is used to control access to a shared resource. A semaphore holds an integer counter and provides two atomic operations: wait and signal.

Auto keyword: auto keyword is used to let the compiler deduce the type of a variable from the initializer. Lambda functions: `([capture] (parameters) -> return type  body )`. Pass by pointer: `([l_ptr = &l])`. Capture all variables by value: `([=] (parameters) -> return type  body )`. Capture all variables by reference: `([&] (parameters) -> return type  body )`. Capture a specific variable by value: `([x] (parameters) -> return type  body )`. Capture a specific variable by reference: `([&x] (parameters) -> return type  body )`.

Std::async: `std::async(std::launch::async, function, args...)` Async tasks are executed in a separate thread.

Std::future: `std::future<T> f = std::async(std::launch::async, function, args...)` Future is a promise to return a value later, a value that is not yet computed.

future.get(): blocks until the future is ready.

Std::promise: `std::promise<T> p` Promise is a container for a future value.

A promise allows you to provide a value once it has been computed.

Without future and promise the value would have to be explicitly protected by a mutex and a condition variable that could be used to wait for the value to be computed.

Std::packaged_task: `std::packaged_task<T> pt(function)` Packaged task is a task that can be executed later.