C & x86 Review

CS5435 - Spring 2024
Marina Bohuk

The C Language

- Barebones language → Very fast, versatile
- Does not come with any “extras” like Java and Python
  - No classes or objects
- Does not do boundary or index checks
- Does not abstract anything away

Pointers

- Store a memory address of an object
  - char *c;
  - int *x;
  - Object *o;

variable	address	memory
	1000	5
x	1016	1000

Arrays

- Arrays are pointers!
  - int intArray[3] = {2, 4, 6};
- The length allocates space, but there is no index checking
- Why do the addresses jump by 4?
- Strings are just arrays

variable	address	memory
intArray	???	1000
	…	
&intArray[0]	1000	2
&intArray[1]	1004	4
&intArray[2]	1008	6
	1012	???

Example program

```c
#include <stdio.h>
#include <string.h>


void greeting( char* temp1 )
{
        char name[400];
        memset(name, 0, 400);
        strcpy(name, temp1);
        printf( "Hi %s \n", name );
}


int main(int argc, char* argv[] )
{
        greeting( argv[1] );
        printf( "Bye %s\n", argv[1] );
}
```

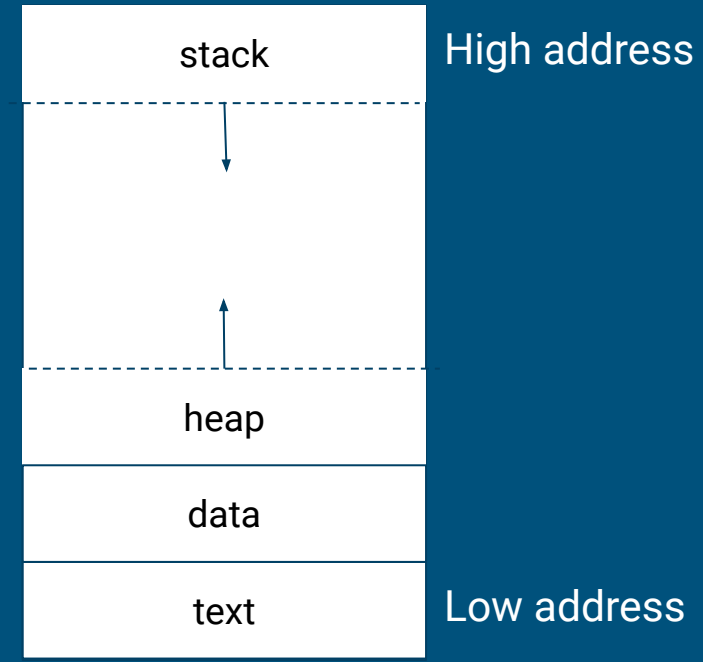
Compiling

gcc -ggdb -mpreferred-stack-boundary=2 -zexecstack -fno-stack-protector -no-pie -fno-pie -m32 meet.c -o meet

- ggdb: debugger symbols
- mpreferred-stack-boundary=2: align the stack in 4 byte chunks
- zexecstack: use an executable stack
- fno-stack-protector: disable stack protections
- no-pie/fno-pie: no ASLR
- m32: use 32-bit x86, not 64

Memory layout

- Each program has its own memory layout including:
  - Stack: stores local variables
  - Heap: dynamic memory for the programmer to allocate
  - Data: stores global variables
  - Text: stores the code being executed
- Each byte of memory is assigned an address



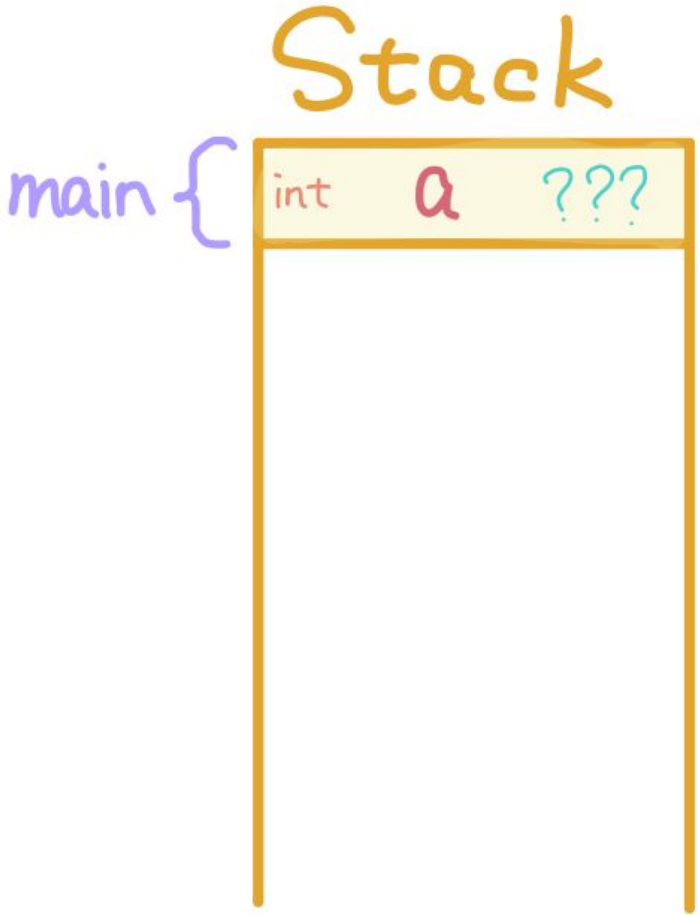
Memory allocation

Static

- Amount of memory needed already known at compile time
  - int array[10];
- This memory is allocated on the **stack**.

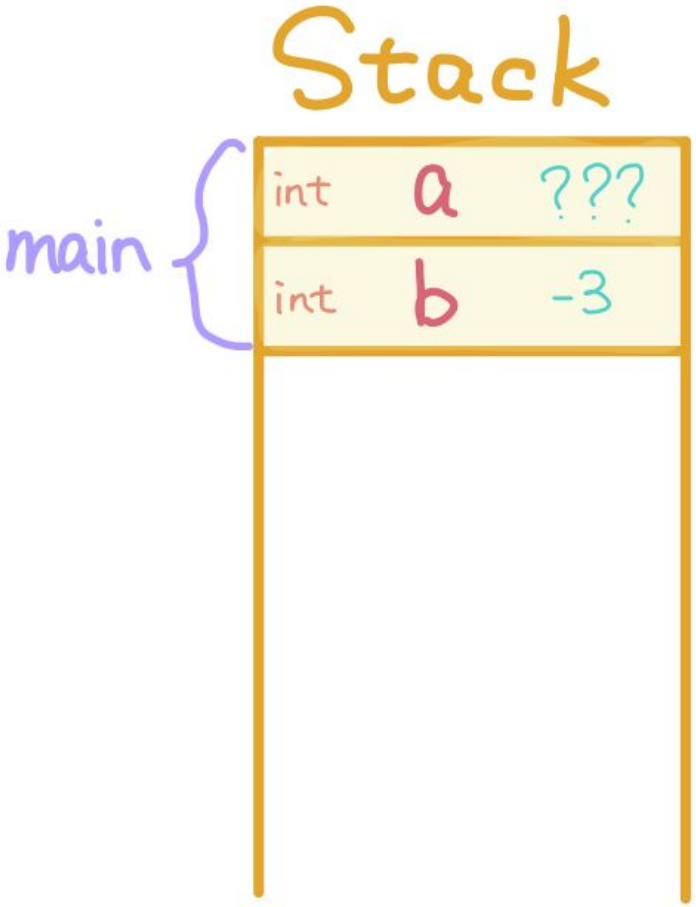
Dynamic

- Amount of memory needed not known
  - int *array = (int*)malloc(n * sizeof(int));
- This memory is allocated on the **heap**.



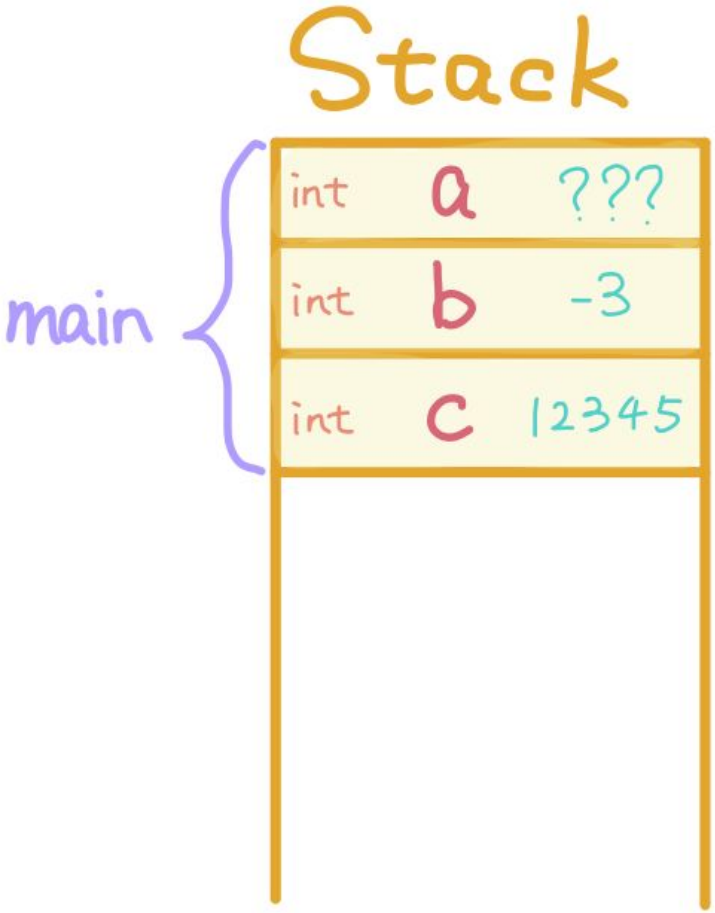

```
int hello() {
  int a = 100;
  return a;
}
int main() {
  int a;
  int b = -3;
  int c = 12345;
  int *p = &b;
  int d = hello();
  return 0;
}
```

source: https://courses.engr.illinois.edu/cs225/fa2022/resources/stack-heap/



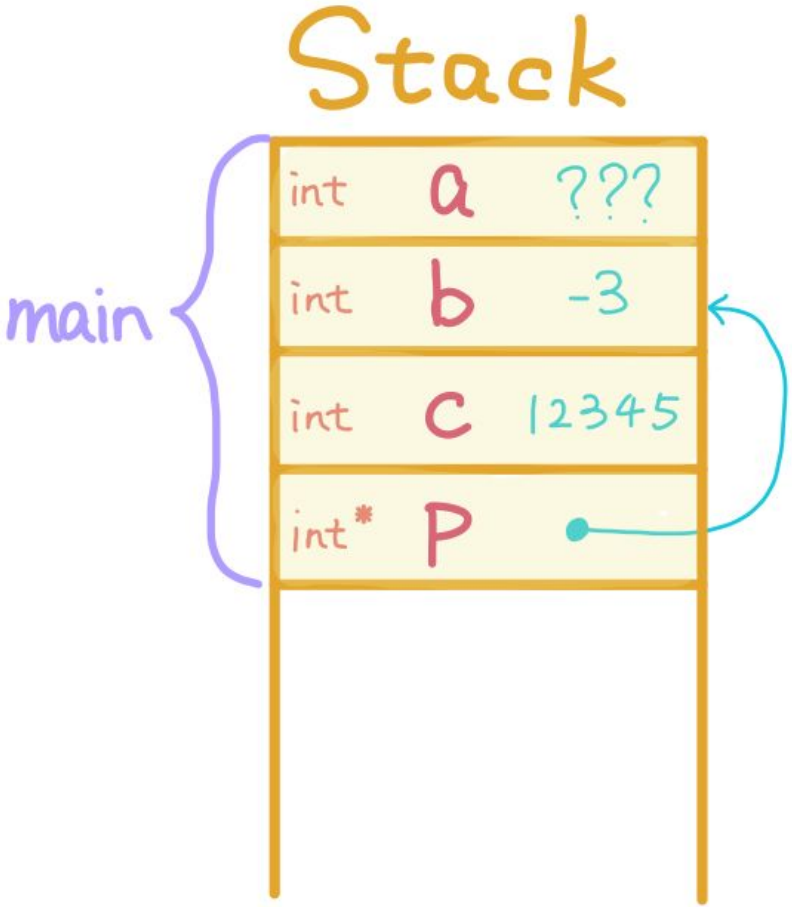

```
int hello() {
  int a = 100;
  return a;
}
int main() {
  int a;
  int b = -3;
  int c = 12345;
  int *p = &b;
  int d = hello();
  return 0;
}
```

source: https://courses.engr.illinois.edu/cs225/fa2022/resources/stack-heap/



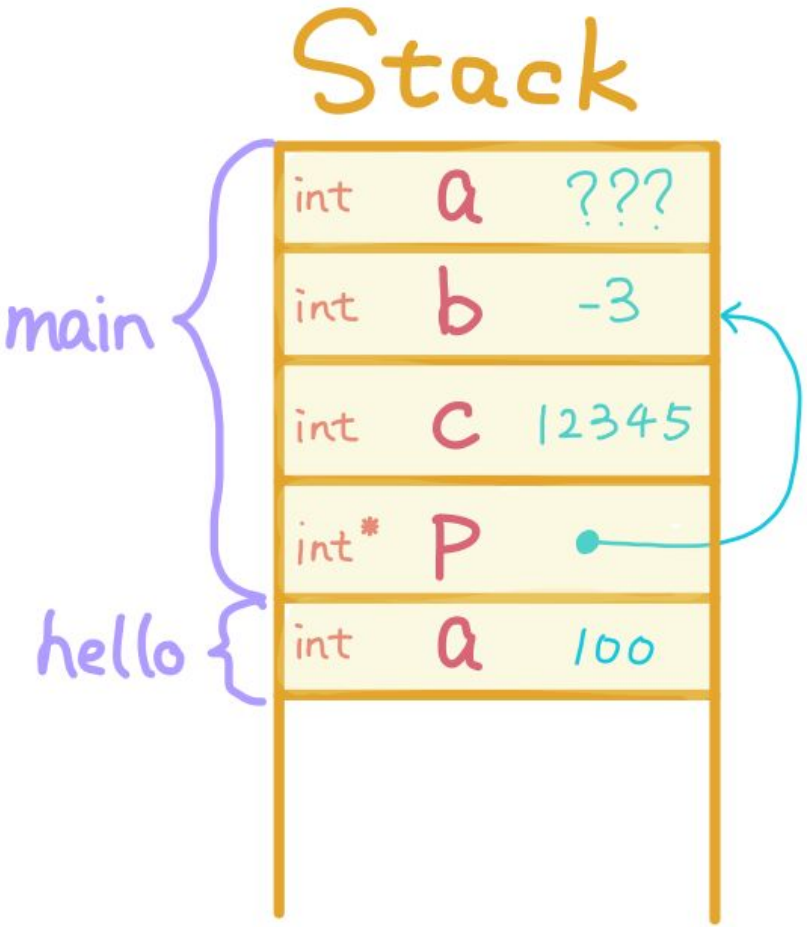

```
int hello() {
  int a = 100;
  return a;
}
int main() {
  int a;
  int b = -3;
  int c = 12345;
  int *p = &b;
  int d = hello();
  return 0;
}
```

source: https://courses.engr.illinois.edu/cs225/fa2022/resources/stack-heap/



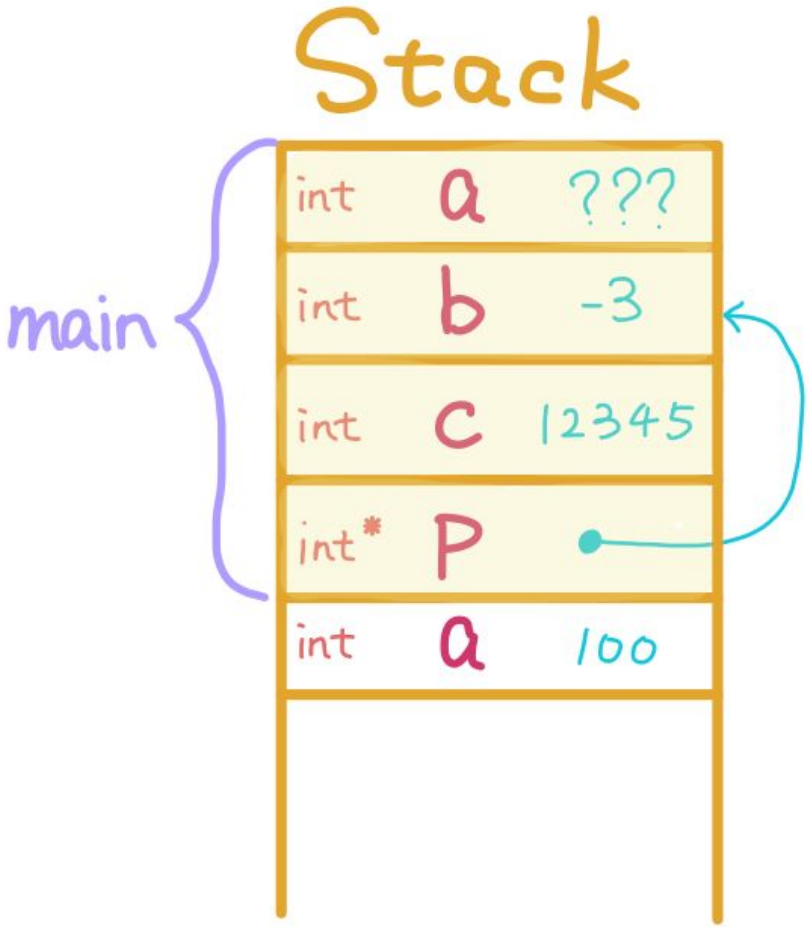

```
int hello() {
  int a = 100;
  return a;
}
int main() {
  int a;
  int b = -3;
  int c = 12345;
  int *p = &b;
  int d = hello();
  return 0;
}
```

source: https://courses.engr.illinois.edu/cs225/fa2022/resources/stack-heap/



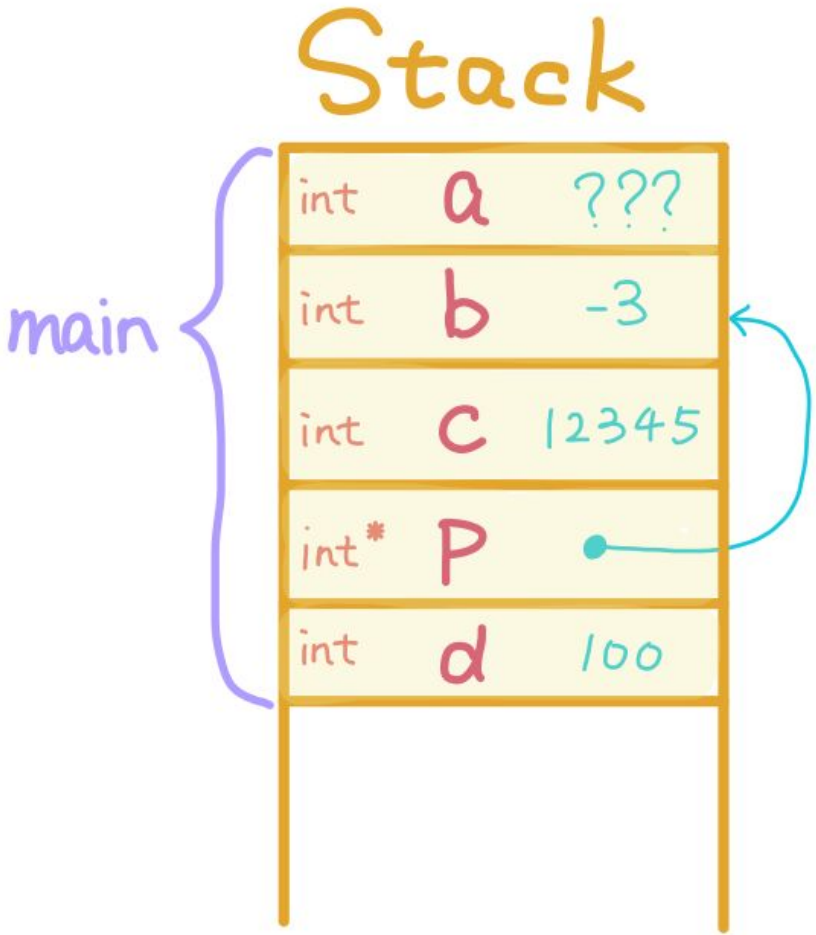

```
int hello() {
  int a = 100;
  return a;
}
int main() {
  int a;
  int b = -3;
  int c = 12345;
  int *p = &b;
  int d = hello();
  return 0;
}
```

source: https://courses.engr.illinois.edu/cs225/fa2022/resources/stack-heap/



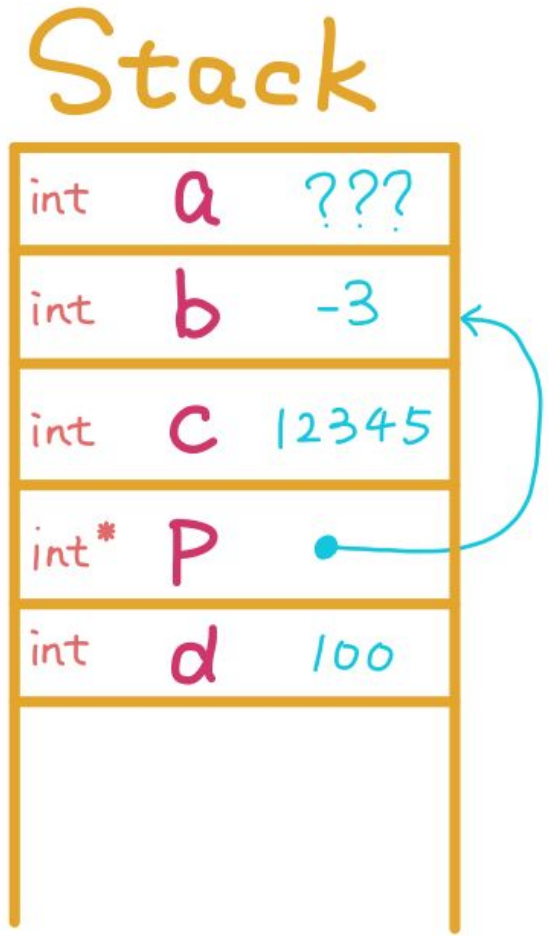

```c
int hello() {
  int a = 100;
  return a;
}
int main() {
  int a;
  int b = -3;
  int c = 12345;
  int *p = &b;
  int d = hello();
  return 0;
}
```

source: https://courses.engr.illinois.edu/cs225/fa2022/resources/stack-heap/




```c
int hello() {
  int a = 100;
  return a;
}
int main() {
  int a;
  int b = -3;
  int c = 12345;
  int *p = &b;
  int d = hello();
  return 0;
}
```

source: https://courses.engr.illinois.edu/cs225/fa2022/resources/stack-heap/

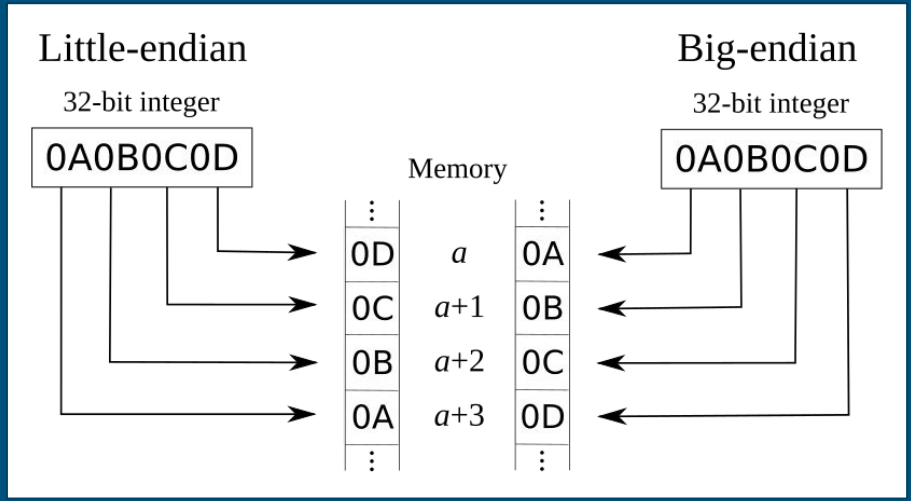



```
int hello() {
  int a = 100;
  return a;
}
int main() {
  int a;
  int b = -3;
  int c = 12345;
  int *p = &b;
  int d = hello();
  return 0;
}
```

source: https://courses.engr.illinois.edu/cs225/fa2022/resources/stack-heap/

x86 assembly

- x86 first came around in the 70s and is still used without much change today
- C is compiled into x86 assembly
- Little endian (important when writing exploits!)
  - Endianness: how to order bytes in a unit
- Uses registers as a quick way to temporarily store information
  - Variables built into the processor
  - eax, ebx, ecx..



Example x86 instructions

mov ecx, eax // move value in eax into ecx

mov eax, [ebx] // move 4 bytes at the memory address pointed to by ebx into eax

add eax, 0x4 // also sub, inc/dec

push eax // push the value in eax onto the stack

pop eax // pop the value on top of the stack into eax

call function

jmp 0xffffffb3 // jump to an address

x86 Calling Convention

A set of rules between functions:

- How are parameters passed? → in registers
- Where are registers preserved? → on the stack
- Where should local vars be stored? → on the stack
- How are values returned? → in register eax

Caller summary

- Prologue: Tasks to take care of BEFORE calling the subroutine
  - Save the caller-save registers
  - Place parameters on the stack (or in registers for 64-bit)
- Call the subroutine
  - The `call` instruction places the return address on the stack
- Epilogue: Tasks AFTER calling the subroutine
  - Remove params from the stack
  - Access return value in eax
  - Restore saved registers

Callee summary

- Prologue: Tasks to perform BEFORE executing the function body
  - Allocate space for local variables (on the stack): `sub 8, %esp`
  - Save callee-save registers
- Function body
- Epilogue: Tasks to perform AFTER executing the function body but BEFORE leaving the function
  - Save return value to eax
  - Restore callee-save registers (pop them from the stack)
  - Deallocate local variables
  - Return

GDB

- A debugger that allows you to see what's going on inside your C program
- PEDA: https://github.com/longld/peda
- Breakpoints:
  - You can set a breakpoint to tell GDB to stop when it reaches a certain location
  - `b hello` // function name
  - `b 123` // line number
- Disassemble:
  - Show the assembly code for a given function
  - `disas main` // function name
- Print out the current stack frame
  - `info frame`
- Print out memory
  - x/64x 0xbffff5cc // print out 64 bytes beginning at address 0xbffff5cc
- Step through code one line at a time
  - `step`

VM for HW 4

ssh your-net-id@35.226.118.201

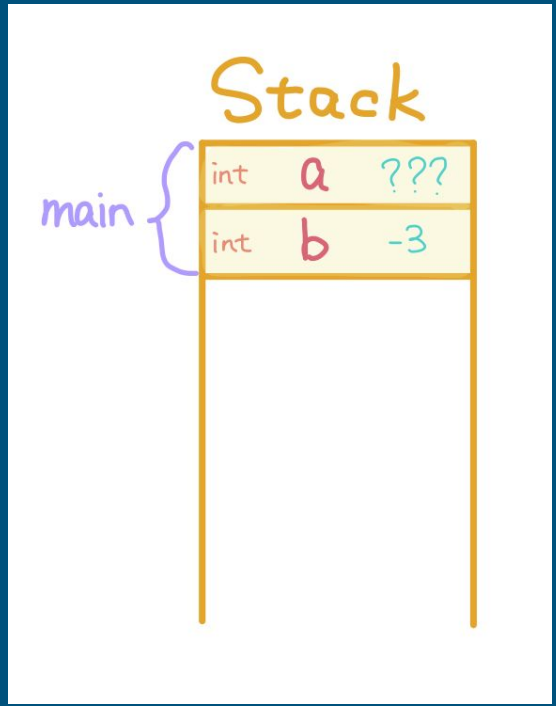
password: your net id (change it as soon as you login)

Simple C program

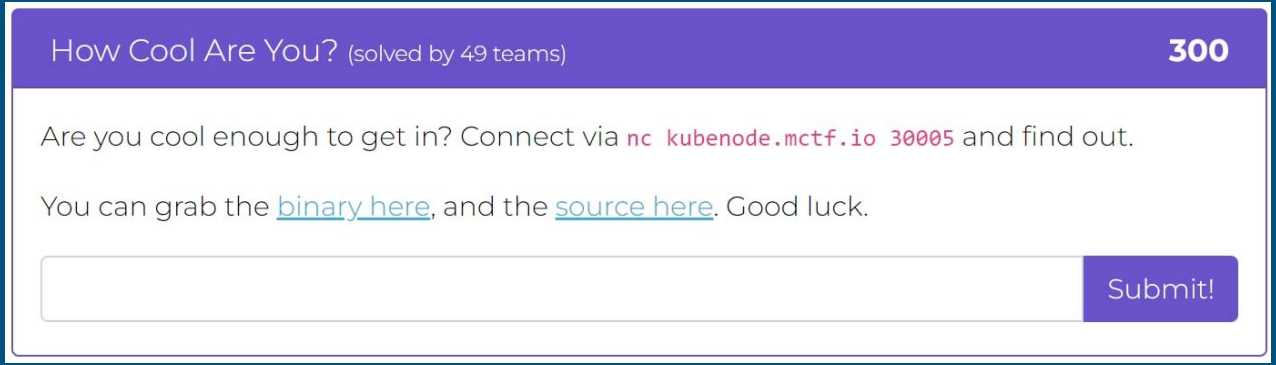
```c
#include <stdio.h>
#include <string.h>


void greeting( char* temp1 )
{
        char name[400];
        memset(name, 0, 400);
        strcpy(name, temp1);
        printf( "Hi %s \n", name );
}


int main(int argc, char* argv[] )
{
        greeting( argv[1] );
        printf( "Bye %s\n", argv[1] );
}
```

Simple Stack Smashing Example

- Preview of stack smashing before Thursday
- Capture the Flag (CTF) Challenge
- Goal: get the flag! (a_flag_looks_like_this)
- Files:
  - Source: https://tinyurl.com/stacksmash5435
  - Binary: https://tinyurl.com/stacksmash5435bin
- Connect to the challenge:
  - `nc kubenode.mctf.io 30005`



Simple Stack Smashing Example



```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>

int main() {
    // Declare variables
    int your_estimated_coolness;
    char your_name[64];

    // Disable buffering for stdin and stdout
    setbuf(stdin, NULL);
    setbuf(stdout, NULL);

    // Seed the random number generator with the current time
    srand(time(NULL));

    // Talk to user
    printf("Whoa there, my dude. Only the truly cool are allowed in here.\n");

    // Use the standard coolness formula to determine how cool the user is
    your_estimated_coolness = 500 + 10 * (rand() % 1000);

    // Read in the user's name
    printf("What's your name? ");
    gets(your_name);

    // Evaluate them
    printf("%s, it looks like your coolness value is %d.\n", your_name, your_estimated_coolness)
    if (your_estimated_coolness < 1500000001) {
        printf("I'm sorry, you're just not cool enough. Get lost!\n");
    } else if (your_estimated_coolness > 1500000001) {
        printf("Whoa, you're... you're actually a bit too cool for us. Sorry...\n");
    } else if (your_estimated_coolness == 1500000001) {
        printf("Wow! Radical... You're in. Have fun.\n");
        printf("Oh, right, here's the flag: %s\n", getenv("FLAG"));
    }

    return 0;
}
```

Source:
https://tinyurl.com/stacksmash5435

Binary:
https://tinyurl.com/stacksmash5435bin

What does this program do?
Where is the vulnerability?

1500000001 == 0x59682F01

More security practice

- MetaCTF:
  - Next monthly Flash CTF: https://mctf.io/apr2024
  - For an int overflow challenge, check out February's Flash CTF
- PicoCTF