Assignment 1 due reminder

Good luck <3

Pointers Recap

– Used to store a memory address

– Can point to any type of data (int, struct, array)

– The data at the address can be accessed (dereference) `*`

– The address at a variable can be retrieved (address_of) `&`

Pointers syntax cheat sheet

- Declare a pointer: `int *int_pointer;`

- Address of: `&my_variable;`

- Dereference (Get the value at a pointer): `*int_pointer;`

Dynamic Arrays and Memory

The Stack

– Where information about your program execution is stored

  – Which functions are called, in what order

  – Which variables are created, and where

– When a block of code is executed `{ }`, a stack frame is created

– When the block is completed, the stack frame is removed (and anything inside it destroyed)

When a stack frame is created, enough memory to store everything in the frame is allocated to the frame

The Stack

```c
int main(void) {
    int user_age = 20;
    int array[5] = {1, 2, 3, 4, 5};

    return 0;
}
```

Stack

main

int: user_age

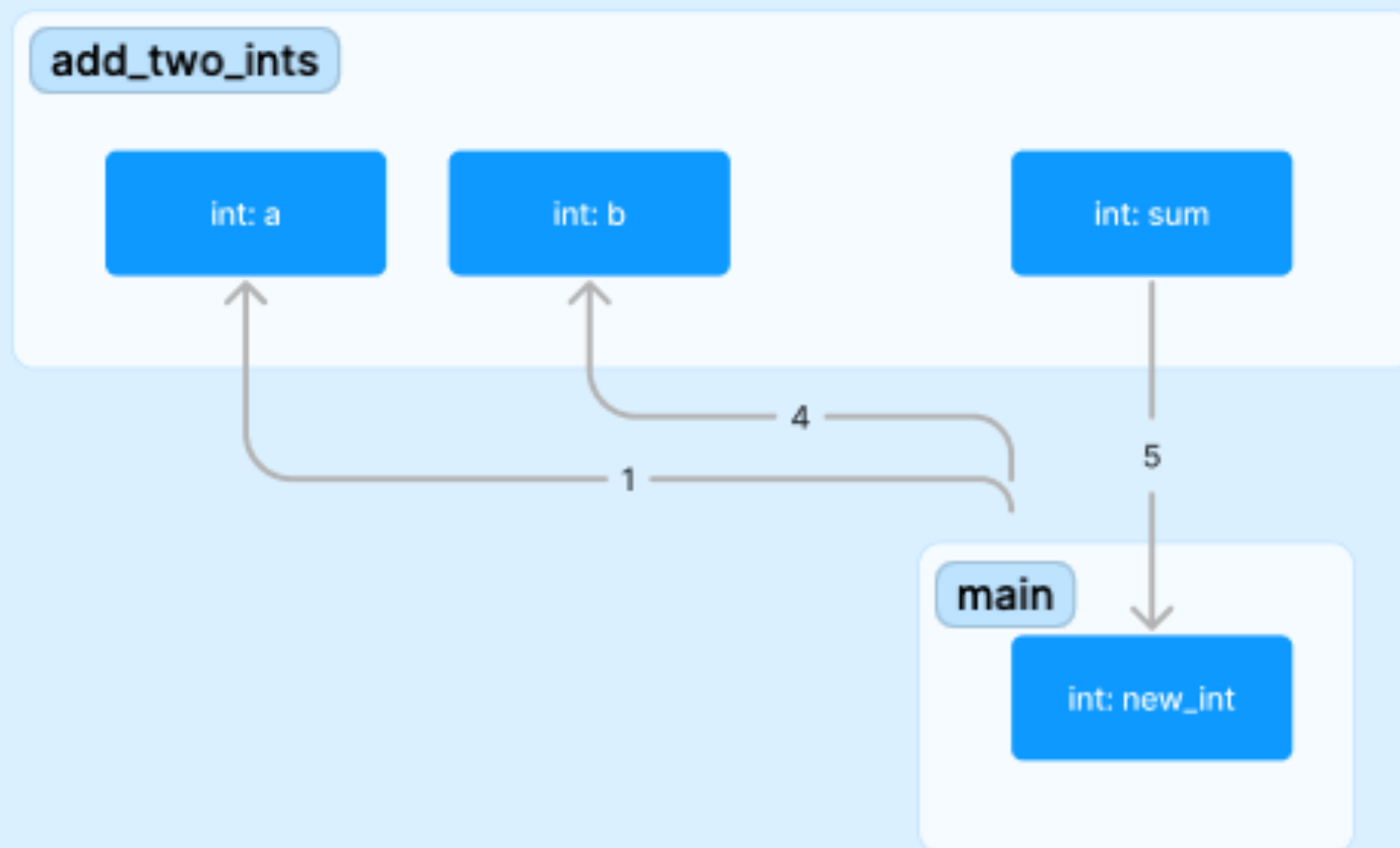int array[5] | 1 | 2 | 3 | 4 | 5

int: return

The Stack

```c
int add_two_ints(int a, int b) {
    int sum = a + b;
    return sum;
}

int main(void) {
    int new_int = add_two_ints(1, 4);

    retrun 0;
}
```

Quick demo + whiteboard

What if we want to create memory with an undetermined size?

– We can't use stack frames... the program needs to know how big the frame is before it creates it

Enter: The Heap

- The Heap is a large block of memory that sits outside the stack

- Unlike the stack, the heap is managed entirely by the programmer (in C)

- Nothing is automatically declared or destroyed in the Heap, you have to manage it all! (This can be dangerous)

- *With great power, comes great responsibility*

Using the heap

C provides us some functions to interact with the heap.

`malloc()`

– `malloc` -> Memory Allocation (allocate memory on the heap)

– Returns a pointer to the location on the heap

– We can decide how large the allocation

Calling `malloc`

- `ptr = (cast-type*) malloc(byte-size)`

Example:

```c
#include <stdio.h>

int main(void) {
    malloc(1000);
    malloc(sizeof(int));
    malloc(sizeof(char) * 50);

    return 0;
}
```

Whiteboard

Dynamic arrays on the heap

A common way of using malloc is to create dynamic arrays:

```c
int main(void) {
    int num_elements;
    scanf("%d", num_elements);

    int *data = malloc(num_elements * sizeof(int));

    return 0;
}
```

Freeing malloc'd data

– This is where the responsibility comes in...

– C automatically frees stack frames after they finish, meaning we don't have to clean up after ourselves.

– We need to manually clean up the heap, otherwise we will cause a memory leak.

free

```c
int *data = malloc(num_elements *
sizeof(int));
...
free(data);
```

`realloc`

– Either resizes the existing allocation (freeing what is no longer needed)

– Or allocates a new allocation

– Always returns the address of the new allocation, even if it's in the same position

realloc

```c
int *data = malloc(num_elements *
sizeof(int));
num_elements += 40;

data = realloc(data, num_elements *
sizeof(int));
...
free(data);
```

Remember, we can treat pointers like arrays

Demo

Feedback

https://forms.office.com/r/K3PjvWebtD

COMP1511 Lecture Feedback