

CSCA48 - Intro to CS II

TA: Angela Zavaleta-Bernuy

Tutorial: T21 (Mondays BV264 10-11am)

Office hours: IC402 on Tuesdays 12-1pm

email: angela.zavaletabernuy@mail.utoronto.ca

website: angelazb.github.io

Week 2 - Jan 20th

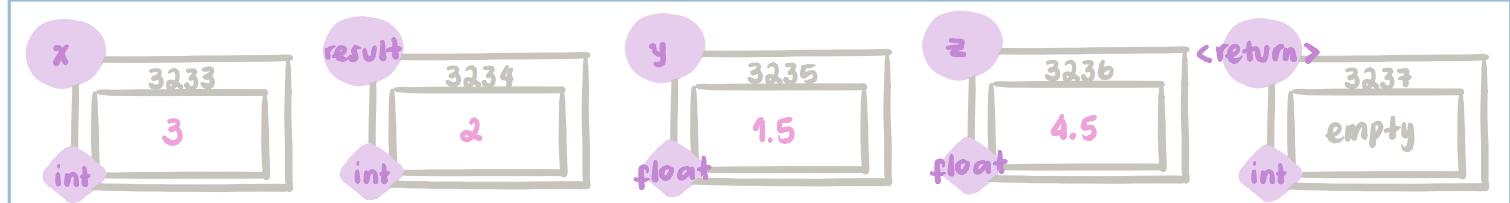
Memory Model in C

- * It's like a locker room! All locker boxes are numbered in increasing order, and only can be accessed by the right user.
- * There are 3 different ways in which a program can get a box in memory:
 - Declaring a variable (each variable gets a box)
 - Input parameters to functions (each gets a box)
 - Return value (gets one box)

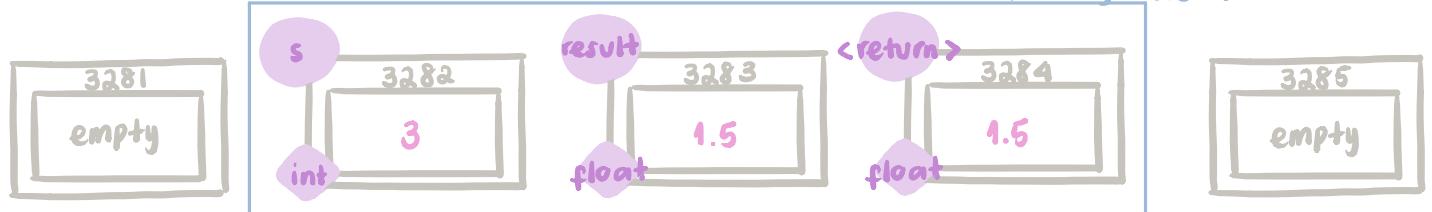
Ex1: Draw the diagram of the memory model right at the point where the result is returned (before the space reserved for the function is released) for the following program.

```
1 #include<stdio.h>
2
3 float div_by_two(int s){
4     float result;
5     result = s / 2;
6     return result;
7 }
8
9 int main(){
10    int x, result;
11    float y, z;
12    x = 3;
13    y = div_by_two(x);
14    z = y + 3;
15    result = z / 2;
16    printf("The result is: %d\n", result);
17 }
```

main()



div-by-two()



Arrays and Strings

* Arrays:

- Collections of contiguous boxes of the same data type. (contiguous in memory)
- Fixed size
- Wrong indexes? You are screwed...

* Strings:

- Arrays of chars.
- End-of-string delimiter '\0'
- Strings are passed to functions by telling it its location in memory, so the function can modify the original input.

Ex2: What do you think this prints out?

```
1 #include<stdio.h>
2
3 int main(){
4     char original[1024] = "This is the original string!";
5     char unoriginal[1024] = "And this is another string!";
6
7     original = unoriginal;
8
9     printf("%s\n", original);
10 }
```

does this even work?! Why?!

How can we copy elements from an array?

Ex3: Does the following code compile? If not, what would you change?

```
1 #include<stdio.h>
2
3 int main(){
4     int array_one[10];
5     int array_two[5];
6
7     for (int i=0; i<5; i++){
8         array_two[i]=i;
9     }
10
11     array_one=array_two;
12 }
```

Nope! C does not allow assignment between arrays like this.

If we need to copy values we need to do it manually.

Ex4: Write a function that takes two input strings (size 10x9) and swaps their content.

Hmm... Interesting!

Week 3 - Jan 27 th

Pointers

- * They are just a variable! With a locker and all, that have the memory address of another variable (which we can decide)
- * When we create a pointer, its type has to match the variable type.

E.g. if you want to initialize a pointer to an int variable: `int *p = NULL;`

* But hold on, how do we use pointers?!

- We first need to assign a variable to our pointer, so we can use `&`.

E.g. Store the address of `x` in `p`: `p = &x;`

- We can also use them to access the value from the locker that they are pointing to.

E.g. Copy the contents of locker (`p`) into `x`: `x = *(p);`
Remember `(p)` stands for the locker # stored in `p`.

How about if we want to access the locker next to `(p)`? `x = *(p+1);`
offset

- Now that we can access contents of the locker stored in the pointer, we can also modify its content.

E.g. Let's change the value of `(p)` to 5 : `*(p) = 5;`

* Don't forget the equivalence between arrays and pointers.

E.g. Store the address of the first element of the array in pointer `p`:
`p = &my_string[0];` or `p = my_string;`

- ALSO, you can use the offset to access the other values of the array.

E.g. If I have an array of 5 elements `p = my_array;`
I can initialize all its values to 0. `*(p) = 0;`
`*(p+1) = 0;` ... and so on.

Ex1 Let's review the 'reverse' function we did in lecture.

`void reverse(char *input, char *output)` make the output be the reverse
of the input strings.

Ex2 Create the function 'reverse_in place' that reverses a string in place
(don't use a temp array)

`void reverse_in place (char *input)` can you modify the existing one?
Hint: use pointers + offset.

Ex3 Given the starter file `ex3.c`, implement the function 'poke Around' to
print the values stored in the other variables around it.

`void poke Around (char *p)` Hint: use pointers and offsets again ☺

Week 4 - Feb 3rd

c - c - d - a - b - c
d - e - d - b - b - b

Compound Data Types (CDT)

- * Useful to represent information about entities that have multiple properties.
- E.g. A student record needs to have fields like name, student number, age, etc.
- * We want to keep all these information together and bundle it up in a single package.
- * How can you define it?

```
typedef struct struct-name
{
    int field-name;
    // More data
} new-type-name;
```

- * How to use them?
 - Declare a variable:
`new-type-name v;`
 - Access a field:
`v.field-name = 5;`
 - Pass them or return them from a function:
`new-type-name update-func(new-type-name v, int value1, ...)`

* How does it look in memory?

- A variable of a CDT gets one locker only!
- Passing or returning a CDT creates a copy
- Using it with pointers.

```
new-type-name v;  
new-type-name *vp;  
vp = &v;  
vp->field-name = 5;
```

Week 5 - Feb 10th

Office Hours Change! Now: Fridays 4-5pm in IC402

Dynamic memory management

Using built-in functions, you can ask for some memory that persists even when the function returns. This memory given to you is stored in a separate area

in memory called the heap.

* To use: call the `malloc()` function to allocate memory, it will return a pointer to the block of memory, which is the only way to access this block in memory.

* Don't forget to empty the memory once you are done using it.

To allocate enough space for N elements of type T:

```
T *allocatedPtr = (T *) malloc(sizeof(T), N);
```

To free the memory:

```
free(allocatedPtr)
```

Ex: Implement a dynamic array for restaurant reviews like you did in lecture with Linked Lists.

Week 7 - Feb 24th

Tutorial Slides:

<https://tinyurl.com/A48Week7>

Week 8 - March 2nd

Tutorial Slides:

<https://tinyurl.com/A48Week8>