# Week 3 - The Memory Model

Daniel Alyoshin

CSCA48 - TUT002

January 23, 2026

# Content Covered so Far

- C programming
- Memory model

# Content Covered so Far - Memory Reservation

- Functions get memory reserved upon function call (not before)

# Content Covered so Far - Memory Reservation

- Functions get memory reserved upon function call (not before)
- Function parameters, local variables *(all variables defined by the function are considered local)*, and return value box get 'a locker'

# Content Covered so Far - Communication Between Functions

1. Parameters are copied from the function call into newly allocated 'lockers' - i.e. **parameters are passed by value** (copies of the original)

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

int main() {
    int a = 5;
    int b = 3
    result = sum(a, b);
    return 0;
}
```

# Content Covered so Far - Communication Between Functions

1. Parameters are copied from the function call into newly allocated 'lockers' - i.e. **parameters are passed by value** (copies of the original)
2. Code in the function is executed

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

int main() {
    int a = 5;
    int b = 3
    result = sum(a, b);
    return 0;
}
```

# Content Covered so Far - Communication Between Functions

1. Parameters are copied from the function call into newly allocated 'lockers' - i.e. **parameters are passed by value** (copies of the original)
2. Code in the function is executed
3. When the return is reached, the return value is copied into the return value box which was previously allocated at the function call

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

int main() {
    int a = 5;
    int b = 3
    result = sum(a, b);
    return 0;
}
```

# Content Covered so Far - Communication Between Functions

1. Parameters are copied from the function call into newly allocated 'lockers' - i.e. **parameters are passed by value** (copies of the original)
2. Code in the function is executed
3. When the return is reached, the return value is copied into the return value box which was previously allocated at the function call
4. The value in the return value box is then **copied** into the specified variable at the function call

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

int main() {
    int a = 5;
    int b = 3
    result = sum(a, b);
    return 0;
}
```

# Content Covered so Far - Communication Between Functions

1. Parameters are copied from the function call into newly allocated 'lockers' - i.e. **parameters are passed by value** (copies of the original)
2. Code in the function is executed
3. When the return is reached, the return value is copied into the return value box which was previously allocated at the function call
4. The value in the return value box is then **copied** into the specified variable at the function call
5. Memory allocated for the function is then released

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

int main() {
    int a = 5;
    int b = 3
    result = sum(a, b);
    return 0;
}
```

# Content Covered so Far - Memory Model Diagrams

- Variables in expected order?

# Content Covered so Far - Memory Model Diagrams

- Variables in expected order?
  1. Function parameters

# Content Covered so Far - Memory Model Diagrams

- Variables in expected order?
  1. Function parameters
  2. Local variables

# Content Covered so Far - Memory Model Diagrams

- Variables in expected order?
  1. Function parameters
  2. Local variables
  3. Return value

# Content Covered so Far - Memory Model Diagrams

- Variables in expected order?
  1. Function parameters
  2. Local variables
  3. Return value
- Boxes tagged correctly?

# Content Covered so Far - Memory Model Diagrams

- Variables in expected order?
  1. Function parameters
  2. Local variables
  3. Return value
- Boxes tagged correctly?
  1. Variable name

# Content Covered so Far - Memory Model Diagrams

- Variables in expected order?
  1. Function parameters
  2. Local variables
  3. Return value
- Boxes tagged correctly?
  1. Variable name
  2. Variable type - remember to append `[]` to arrays and $*$ to pointers

# Content Covered so Far - Memory Model Diagrams

- Variables in expected order?
  1. Function parameters
  2. Local variables
  3. Return value
- Boxes tagged correctly?
  1. Variable name
  2. Variable type - remember to append `[]` to arrays and $*$ to pointers
- Are your lockers **for a given function** consecutive? (no empty boxes)

# Content Covered so Far - Memory Model Diagrams

- Variables in expected order?
  1. Function parameters
  2. Local variables
  3. Return value
- Boxes tagged correctly?
  1. Variable name
  2. Variable type - remember to append `[]` to arrays and `*` to pointers
- Are your lockers **for a given function** consecutive? (no empty boxes)
- Are uninitialized variables set to `junk`?

# Your Task for Today

```
float convert_to_degrees(float angle)
{
    // This function converts an angle given in radians to
        degrees
    return angle * 360.0 / (2.0 * 3.1415926535);
}
int main()
{
    int x;
    float ang;
    x = 2;
    ang = convert_to_degrees(x);
    printf("%f\n", ang); // <-- stop just before the printf call
    return 0;
}
```

# Bonus Task

```c
void hard_working_function(int x)
{
    x = x * x;
    printf("x = %d\n", x);
    x = x / x;
    printf("x = %d\n", x);
    x = 3 * x;
    printf("x = %d\n", x);
    x = x + 71;
    printf("x = %d\n", x);
    x = x / 2;
    printf("x = %d\n", x);
}

int main()
{
    int x;
    x = 1;
    hard_working_function(x);
    printf("Actually, x = %d\n", x);
    return 0;
}
```

# Reminders

- Each student to submit their own file (picture or scan of your work) which can be the same as that submitted by members of their group.
- Ensure your file is named **exactly** as instructed on Quercus.
- You are **not** supposed to keep working on this, you will not be marked on correctness so as long as you submit work that shows you were engaged during this tutorial you will get full marks.

# Our Site!

**csca48.alyoshin.dev**