Week 3 Lecture 1 Procedures and functions

Week 2 recap

Nested loops

- Simply, a while loop within a while loop
- Useful for 2-dimensional data (like grids)

```
   col
row 1 2 3 4 5
    1 2 3 4 5
    1 2 3 4 5
    1 2 3 4 5
    1 2 3 4 5
```

```c
#include <stdio.h>

#define ROWS 5
#define COLUMNS 5

int main() {
    int i = 0;

    while (i < ROWS) {
        int j = 1;
        while (j <= COLUMNS) {
            printf("%d ", j);
            j++;
        }
        printf("\n");
        i++;
    }

    return 0;
}
```

structs

- A defined structure of data types, each accessible
- Memory is set aside for each field in each struct
- Useful for assigning a variable to an organised record of data

```
struct pokemon {
    int hp;
    double weight
};
```

enums

- A possible set of values
- Useful for creating labels in your code

```
enum
elemental_type {
FIRE, WATER,
GRASS, DARK };
```

Functions

- So far, you have used functions in your code
- Examples include `printf`, `scanf`, `main` ...
- What actually are these?

Functions

- Functions are reusable blocks of code
- Functions (may) have:
  - **input** (parameters)
  - **actions** (side effects)
  - **output** (results)

Functions

- We **call** functions to execute their body, providing any input necessary
- We can access the result of the function
- We can call a function from anywhere in our programs

Function definition example

```
int add(int x, int y) {
    return x + y;
}
```

- `int ...` -> return type (what **type** should the result be
- `add` -> the name of the function
- `(int x, int y)` -> the **parameters**, what sequence and type of input must be passed in?
- `return` -> evaluate the expression and return the result

Function call syntax

```
add(2, 5);
```

- After we define functions, we want to use them
- The `()` after the name of the function means **call**
- We must pass in the correct sequence of **arguments** of the correct type (`int add` required two integers).

Function calling

We can pass in variables too

```
// A simple function which accepts
two integers (x, y),
// and returns the result (int) of
adding them.
int add(int x, int y) {
    return x + y;
}

int main(void) {
    int year_born = 1994;
    int age = 29;

    add(year_born, age);
}
```

Retrieving the result of a function

```
// A simple function which
accepts two integers (x, y),
// and returns the result (int)
of adding them.
int add(int x, int y) {
    return x + y;
}

int main(void) {
    int year_born = 1994;
    int age = 29;

    int age = add(year_born,
age);
}
```

DEMO

Functions terminology

- **return type** -> the type of data returned by the function
- **result** -> the actual value returned from a function call
- **parameters** -> the type, and sequence of data to be passed into a function (the placeholders)
- **argument** -> the actual value passed into a function's parameters when called
- **return** -> the keyword used to end a function and return the result following

Procedures

not a *real* thing in C, but a useful way to think about some types and roles of functions

Procedures

- Not all functions have to return a result
- We call these *void functions*, or *procedures*
- Procedures **do** something, but don't have a result
- procedures (usually) have a side-effect

procedures	functions
`shut_door`	`check_door_shut`
side effect?	side effect?
result?	result?

procedure syntax

```
void check_door_shut() {

}
```

- This is a function which returns nothing (void)
- We could call this a procedure

Order matters

Functions/procedures have to be defined before they care called

- we can get around this with *function prototypes*
- Place `int add(int x, int y);` at the top of your file to define the int add function for later use

When writing functions in your program, think:

- What **must** I give this function so it can do its job?
- What should it be named?
- What should it return back to me to achieve its goal? (If anything).
- Am I re-writing code that could be turned into a reusable function?

Functions are very important

- They change how we think about code
- When you come across useful, repeatable functionality - make it a function

0, 1, ∞

Feedback

https://forms.office.com/r/Ze4admEWnR