COMP26020 Programming Languages and Paradigms -- Part 1

---

# Custom Types and Data Structures in C

# Custom Types

- Use `typedef` to alias a type

```c
#include <stdio.h>

// 'my_int' is now equivalent to 'long long unsigned int'
typedef long long unsigned int my_int;

int main(int argc, char **argv) {
    my_int x = 12;

    printf("x is: %llu\n", x);

    return 0;
}
```

08-c-custom-types/typedef.c

# Custom Data Structures

- Use `struct` to define a data structure with named fields
- Use `struct <struct name>` to refer to it as a type
- Access the fields of a variable with `<variable_name>.<field_name>`

```c
#include <stdio.h>
#include <string.h> // needed for strcpy

// Definition of the struct:
struct person {
    char name[10];
    float size_in_meters;
    int weight_in_grams;
};

void print_person(struct person p) {
    /* Access fields with '.' */
    printf("%s has a size of %f meters and "
        "weights %d grams\n", p.name,
        p.size_in_meters, p.weight_in_grams);
}
```

```c
int main(int argc, char **argv) {
    // declare a variable of the struct type:
    struct person p1;

    // sets the variable field
    p1.size_in_meters = 1.6;
    p1.weight_in_grams = 60000;
    strcpy(p1.name, "Julie");

    struct person p2 = {"George", 1.8, 70000};

    print_person(p1);
    print_person(p2);
    return 0;
}
```

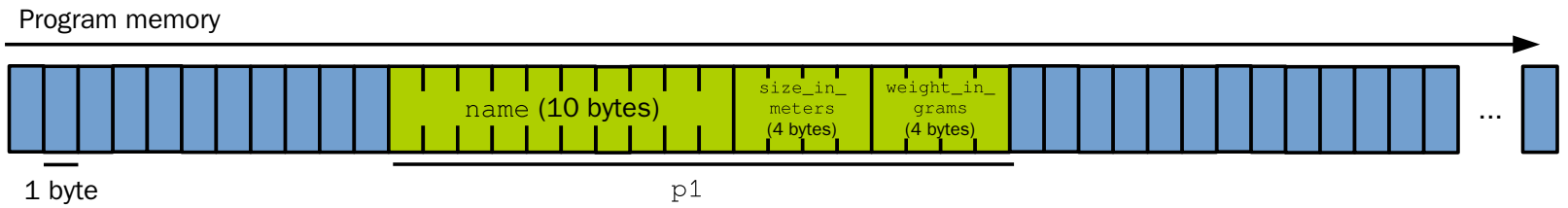
08-c-custom-types/struct.c

# Custom Data Structures

```
struct person {
    char name[10];
    float size_in_meters;
    int weight_in_grams;
};

struct person p1;
```

- Fields are place consecutively in memory (compiler may insert padding):



# Custom Data Structures

- Structures can be typedef'd

```c
#include <stdio.h>
#include <string.h> // needed for strcpy

struct s_person {
    /* fields declaration ... */
};

typedef struct s_person person;

void print_person(person p) { /* ... */}

int main(int argc, char **argv) {
    person p1;
    person p2 = {"George", 1.8, 70000};
    /* ... */
}
```

- Faster:

```c
typedef struct {
    /* fields declaration ... */
} person;
```

# Enumeration

- User defined type for assigning textual names to integer constants

```c
enum color {
    RED,
    BLUE,
    GREEN
};

int main(int argc, char **argv) {
    enum color c1 = BLUE;

    switch(c1) {
        case RED:
            printf("c1 is red\n"); break;
        case BLUE:
            printf("c1 is blue\n"); break;
        case GREEN:
            printf("c1 is green\n"); break;
        default:
            printf("Unknown color\n");
    }

    return 0;
}
```

08-c-custom-types/enum.c

- Under the hood the compiler assigns an integer constant to each value of an enum:

```c
printf("BLUE was assigned: %d\n", BLUE);
printf("GREEN was assigned: %d\n", GREEN);
printf("RED was assigned: %d\n", RED);
```

# Enumeration

- Enumerations can be typedef'd

```
enum {
    BLUE,
    /* ,,, */
} e_color;

enum e_color c1 = BLUE; // without typedef

typedef enum e_color color;

color c2 = RED;
```

- All in once:

```
typedef enum {
    BLUE,
    /* ... */
} color;
```

08-c-custom-types/enum-color.c

# Summary

- Typedef
- Structs
- Enum

---

Feedback form: https://bit.ly/3iwfECU