CMPT 125

Introduction to Computing Science and Programming II

September 22, 2021

stdbool.h

stdbool.h

Write a function that gets an array of ints and a number and checks if it is contained in the array

```
#include <stdbool.h>

bool contains(const int* array, int len, int elt) {
        bool found = false;
        int i = 0;
        while (i < len && !found)     {
                if (array[i] == elt)
                        found = true;
                i++;
        }
        return found;
}
```

found == false

Enum/Typedef/Struct

Enum

User defined data types. Mainly used to assign names to integers.

Examples:

```
enum suit {Hearts, Spades, Clubs, Diamonds};
        // default values are assigned starting from 0
        // i.e., Hearts = 0, Spades = 1, Clubs = 2, Diamonds = 3;


enum emphasis {Bold = 1, Italic = 2, Underline = 4};
        // can define integer values of the names
```

Usage:

```
enum siut card = Spades; // variable of type enum suit
```

The name of the type is enum suit

Typedef

Typedef is used to give a name to a data type.

Examples:

1. typedef int whole_number;
2. enum months {January, February,...};
   typedef enum months month;
3. typedef enum boolean_values {false, true} bool; // all in one line

- Usage:

```
whole_number amount = 23;
bool flag = true;
month my_month = January;
```

Typedef

- Typedef is used to give a name to a data type.
- Typically we define new types outside of all functions.
- This allows the types to be used everywhere in the program
- More examples in types.c

Struct

- So far we have considered only simple types of variables (int, float, char, pointers).
- What if we want a more complicated data type?
- <u>Example</u>:

  A student has:

  First name
  Last name
  ID
  List of grades in homeworks

- We want an array of students.

Hard to keep track of all the information in different arrays.

- We can have array of first names, array of last names, array of IDs...

Struct

```
struct student_info {
        char* first_name;
        char* last_name;
        int ID;
        int grades[5];
};

struct student_info var_student;

typedef struct student_info student;

student list_of_students[180];

list_of_students[10].first_name = "Jack";
...
```

The type is called
struct student_info

Same as struct student_info

Struct

Could also write:

```
typedef struct student_info {
        char* first_name;
        char* last_name;
        int ID;
        int grades[5];
}  student;

student list_of_students[180];

list_of_students[10].first_name = "Jack";
...
```

Struct

Using pointers with structs:

```
student clark;
clark.first_name = "Clark";
```

```
student* student_ptr = &clark;
(*student_ptr).last_name = "Kent"; // accessing a field in struct
student_ptr->id = 123; // accessing a field in pointer to struct
```

A bit more on syntax:
Return values and conditions

Return values and conditions

- All command in C return a value. (Exception: void functions)
- Examples:

```
printf("%d\n" , 3 < 5); // prints 1
printf("%d\n" , 3 == 5); // prints 0
```

<u>if statements</u>:

```
if (cond)
    do_something();
else
    do_something_else();
```

Equivalent to:
if (cond != 0)

<u>while statements</u>:

```
while (cond)
    do_something();
```

Equivalent to:
while (cond != 0)

Multiple conditions

AND of conditions:

```
if (cond1 && cond2)
    do_something();
else
    do_something_else();
```

&& - and
- checks first if cond1 is satisfied
  (i.e., cond1 !=0)
- runs cond2 **only** if cond1 is satisfied

OR of conditions:

```
while (cond1 || cond2)
    do_something();
```

|| - or
- checks first if cond1 is satisfied
  (i.e., cond1 !=0)
- runs cond2 **only** if cond1 is not satisfied

Recall the example: while (i < len && !found)

Global variables
vs
Static variables

Global variables

So far we have seen only variables defined inside functions.
The scope of the variables is limited only to the function.

It is possible to define a **<u>global variable</u>** that is visible everywhere.

```
#include <stdio.h>

int counter = 0; // init the global variable to zero

int main() {
        printf("global counter %d\n", counter); // prints 0
        counter++;
        printf("global counter %d\n", counter); // prints 1
        int counter = 0; // local variable
        printf(“local counter %d\n", counter); // prints 0
        return 0;
}
```

Static variables

It is also possible to define a **static variable** that will "remember“ its value in different calls of the function.

```
#include <stdio.h>

void test_static_count() {
        static int count = 0; // initialized only once!
        // do something...
        count++;
        printf("count = %d\n", count);
}

int main(void) {
        test_static_count(); // prints “count = 1”
        test_static_count(); // prints “count = 2”
        test_static_count(); // prints “count = 3”
        …
```

Macros

Using macros: #define

#define creates a constant that can be use globally.

Macros are not variables. Cannot be changed by the program.

```
#include <stdio.h>

#define COURSE_NAME "CMPT125"
#define PI 3.1415925

int main() {
        printf("%f\n", PI); // prints 3.1415925
        printf("%s\n", COURSE_NAME); // prints CMPT125
        printf("PI\n"); // prints PI
        …
```

Using macros: #define

#define macros are simply textual substitutions.

Preprocessor replaces the occurrences of the macros before compiling the code.

```
#include <stdio.h>

#define MY_FRAC 70/14
#define SQR(a) a*a

int main() {
        float x = MY_FRAC; // replaced by float x = 70/14;
        int y = SQR(5); // replaced by int y = 5*5;
        int z = SQR((5+2)); // replaced by int z = (5+2)*(5+2);
        int w = SQR(5+2); // replaced by int z = 5+2*5+2;
        …
```

Type casting in C

Type casting

- C allows conversions between different types of variable.
- Done when one data type can be changed to a different data type.
- Example:
  - int to float
  - short to int
  - int to long
- We can also type cast the result to make it of a particular data type.
- Need to be careful. Information may be lost!
- Examples:
  - float to int
  - int to char
  - char* to int*

Questions?

Comments?