Today’s Goals

- Functions
  - Definition
  - Return
  - Arguments
- Prototypes
- Locals and Globals
- static

Functions

- Function: Unit of operation
  - A series of statements grouped together
- Must have the main function
- C functions are stand-alone
- Most programs contain multiple function definitions
  - Must be declared/defined before being used

Identify Repeated Code

```c
int main() {  
    int choice;  
    printf("=== Expert System ===\n");  
    printf("Question1: ...\n");  
    printf("1. Yes\n");  
    printf("0. No\n");  
    printf("Enter the number corresponding to your choice: ");  
    scanf("%d", &choice);  
    if (choice == 1) { /* yes */  
        printf("Question 2: ...
");  
        printf("1. Yes
");  
        printf("0. No
");  
        printf("Enter the number corresponding to your choice: ");  
        scanf("%d", &choice);  
        /* skipped */
    }
}
```

Identify Similar Code

```c
int menuChoice() {  
    int choice;  
    printf("1. Yes
");  
    printf("0. No
");  
    printf("Enter the number corresponding to your choice: ");  
    scanf("%d", &choice);  
    return choice;
}
```

```c
int main() {  
    int choice;  
    printf("=== Expert System ===\n");  
    printf("Question1: ...
");  
    choice = menuChoice();  
    if (choice == 1) { /* yes */  
        printf("Question 2: ...
");  
        choice = menuChoice();  
        /* skipped */
    }  
}
```

Use Parameters to Customize

```c
void km_mile_conv(int choice) {  
    int input;  
    printf("Enter a %s value: ", choice==1?"mile":"km");  
    scanf("%lf", &input);  
    if (choice == 1)  
        printf("%f mile(s) = %f km(s)
", input, input*1.6);  
    else  
        printf("%f km(s) = %f mile(s)
", input, input/1.6);  
}
```

```c
int main() {  
    int choice;  
    scanf("%d", &choice);  
    switch (choice) {  
        case 1:  
            km_mile_conv(choice);  
            break;  
        case 2:  
            km_mile_conv(choice);  
            break;  
        /* more cases */
    }  
}
```
Function-oriented

- C came before OO concept
- C program resemble java programs with a single giant class
- C is procedural
  - Program organization and modularization is achieved through function design
  - Carefully plan your function return type and parameter list
  - Write small functions!

Function Call

```c
void km_to_mile() {
    printf("Enter a mile value: ");
    scanf("%lf", &mile);
    km = mile * 1.6;
    printf("%f mile(s) = %f km\n", mile, km);
}
int main() {
    km_to_mile();
    km_to_mile();
    return 0;
}
```

Function Return and Parameters

- The syntax for C functions is the same as Java methods
- `void` keyword can be omitted

```c
void km_to_mile(void) {
    mile_to_km();
}
int main() {
    int choice;
}
```

Use of `return` in `void` Functions

- Exit from the function

```c
void getinput() {
    int choice;
    while (1) {
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                /* some action */
                break;
            case 0:
                return; /* exit from getinput */
        }
    }
}
```

Function Prototype

- A prototype is a function declaration which includes the return type and a list of parameters
- A way to move function definitions after `main`
- Need not name formal parameters

```c
/* function prototypes */
double km2mile(double);
double mile2km(double);
int main() {
    double miles2km(double m) {
    double km2mile(double k) {
}
```

Local/Global Variables

- Variables declared inside a function are local
- Function arguments are local to the function passed to
  - A global variable is a variable declared outside of any function.
- In a name conflict, the local variable takes precedence
- When local variable shadows function parameter?
Scope of Global Variables

- The scope of a global variable starts at the point of its definition.

- **Globals should be used with caution**
  - Avoid changing a global inside a function
  - Change a global by setting it the return value of a function
  - If using globals at all, declare them at the top.

```c
int x;
int f() {
    int y;
    int g();
    int main() {
    }
}
```

Call by Value

- Same as Java, modification to function arguments during function execution has no effect outside of function

```c
void f(int x) {
    x = x * x;
    printf("%d", x);
}

int main() {
    int x = 3;
    f(x);          // Does not change the value of x in main.
    printf("%d", x);
    return 0;
}
```

Storage Classes

- **auto**
  - The default – life time is the defining function
  - De-allocated once function exits

- **static** (w.r.t. local variables)
  - Life time is the entire program – defined and initialized the first time function is called only
  - Scope remains the same

```c
void f() {
    static int counter = 0;
    counter++;
}
```

static: globals and functions

- Using the keyword `static` in front of a global or a function changes the linkage, that is, the scope across multiple files.

- **static** changes the linkage of an identifier to internal, which means shared within a single (the current) file

- We will discuss more of linkage and related keywords, as well as header files when we discuss multiple source files

Documenting Functions

- A comment for each function
- Use descriptive function name, parameter names

```c
#include <stdio.h>
#include <math.h>
/* truncate a value to specific precision */
double truncate(double val, int precision) {
    double tmp = pow(10, precision);
    tmp = (int) (val / tmp);
    return tmp / adj;
}
int main() {
}
```

Keep main Uncluttered

- Your `main` function should consist mainly of function calls
- One main input loop or conditional is okay
- Write your `main` and choose your function name in such a way so that
  - the main algorithm and program structure is clearly represented
  - the reader can get an idea how your program works simply by glancing at your `main`
Summary

• Use functions to modularize your code
• Your `main` should include mostly function calls
• Learn to use prototypes
• Learn the difference between C and Java’s `static` keyword