Today's Goals

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

Section 1

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

```
int main() {
  int choice;

  printf("=== Expert System ===\n");
  printf("Question1: ...\n");
  printf(
    "1. Yes\n"
    "0. No\n"
    "Enter the number corresponding to your choice: ");
  scanf("%d", &choice);

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


Identify Repeated Code

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

int main() {
  int choice;

  printf("=== Expert System ===\n");
  printf("Question1: ...\n");
  choice = menuChoice();

  if (choice == 1) { /* yes */
    printf("Question 2: ...\n");
  choice = menuChoice();
  /* skipped */
```


Identify Similar Code

```
int main() {
  int choice;  double km, mile;
  scanf("%d", &choice);

  switch (choice) {
  case 1:
    printf("Enter a mile value: ");
    scanf("%lf", &mile);
    km = mile * 1.6;
    printf("%f mile(s) = %f km\n", mile, km);
    break;

  caes 2:
    printf("Enter a km value: ");
    scanf("%lf", &km);
    mile = km / 1.6;
    printf("%f km = %f mile(s)\n", km, mile);
    break;

  default:
    printf("\n*** error: invalid choice ***\n");
  }
}
```

Similar unit

Similar unit

Use Parameters to Customize

```
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)\n", input, input*1.6);
  else
    printf("%f km(s) = %f mile(s)\n", input, input/1.6);
}
int main() {
  int choice;
  scanf("%d", &choice);
  switch (choice) {
  case 1:
    km_mile_conv(choice);
    break;
  caea 2:
    km_mile_conv(choice);
    break;
  /* more cases */
  }
}
```

More readable **`main`**

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

```
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

```
void km_to_mile(void) {

}

mile_to_km() {

}

int main() {
  int choice;
}
```


Use of **`return`** in **`void`** Functions

- Exit from the function

```
void getinput() {
  int choice;

  while (1) {
    scanf("%d", &choice);

    switch (choice) {
    case 1:
      /* some action */
      break;
    case 0:
      return; /* exit from getinput */
    }
  }
}
```


Section 2

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

```
/* function prototypes */
double km2mile(double);
double mile2km(double);
int main() {
}
/* actual function definitions */
double km2mile(double k) {

}
double mile2km(double m) {

}
```


Section 3

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?

```
int x = 0;
int f(int x) {
  int x = 1;
  return x;
}

int main() {
  int x;
  x = f(2);
}
```

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.

```
int x;
int f() {
}

int y;
int g(){
}

int main() {

}
```


Section 4

Call by Value

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

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

int main() {
  int x = 3;
  f(x);
  printf("%d", x);
  return 0;
}
```

The variable `x` in `f` gets a *copy* of the value of the variable `x` in `main`.

Does not change the value of `x` in `main`.

Example

```
int foo = 2;
int f(int bar) {
  bar += foo;
  foo = bar+3;
  printf("%d, %d\n", foo, bar);
  return bar;
}
int g(int bar) {
  int foo = 5;
  foo += bar;
  bar = foo++;
  printf("%d, %d\n", foo, bar);
  return (foo*2);
}
```


Example

```
int main () {
  int bar = 3;

  printf("%d\n", g(bar));
  printf("%d, %d\n", foo, bar);
  printf("%d\n", g(f(foo)));
  return 0;
}
```


Section 5

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

```
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

```
#include <stdio.h>
#include <math.h>

/* truncate a value to specific precision */
double truncate(double val, int precision) {
  double adj = pow(10, precision);
  int tmp;

  tmp = (int) (val * adj);
  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