CIS 314
Conditional, Functions, Recursions

Prof. Michel A. Kinsky
Conditional execution

• Conditional constructs provide the ability to control whether a statement list is executed

• If statement
  ‣ If
    • if (Expression)
    • Action

Expression

true
false

Action
Conditional execution

• Conditional constructs provide the ability to control whether a statement list is executed

• If statement
  - If
    - if (Expression)
      - Action

```c
int num1;
int num2;
printf("Enter two integers: ");
scanf("%d %d", &num1,&num2);
if (num1 > num2) {
    int remember_num1 = num1;
    num1 = num2;
    num2 = remember_num1;
}
printf("Inputs in sorted order: %d \t %d", num1 , num2 );
```
Conditional execution

Conditional constructs provide the ability to control whether a statement list is executed.

If statement
  - if-else
    - if (Expression)
      - true: Action\(_1\)
      - false: else Action\(_2\)
Conditional execution

- Conditional constructs provide the ability to control whether a statement list is executed

- **If statement**
  - if-else
    - if (Expression)
      - Action 1
    - else
      - Action 2

```c
int num1;
int num2;
printf("Enter two integers: ");
scanf("%d %d", &num1, &num2);

int max;
if (num1 > num2) {
    max = num1;
} else {
    Max = num2
}

printf("Maximum number is: %d", max);
```
Conditional execution

- For loop
- While loop
- Do while loop
• For loops

  ▶ The general format when using for loops is

  • for ( initialization; loopContinuationTest; increment )

  Initialize variable

  Condition
  Test the variable

  Statement

  Increment variable

  Initialize variable

  true

  statement

  false
Iteration

• For loops
  ‣ The general format when using for loops is
    • for ( initialization; loopContinuationTest; increment )
      Statement

```c
int counter;
for(counter = 1; counter <= 10; counter++)
    printf("Current counter value \%d\n", counter);
```
Iteration

• While loop
  ‣ while loop repeated until condition becomes false
    • Initialization;

      while (loopContinuationTest){
          statement
          increment;
      }

}
• While loop
  ‣ while loop repeated until condition becomes false
  
  • Initialization;

  while (loopContinuationTest){
    statement
    increment;
  }

  int counter =1;      //initialization
  while (counter <= 10){ //repetition
    condition
    printf("Current counter value %d
\n", counter);
    counter++;             //increment
  }

  false

  true
Iteration

• Do while
  ‣ do/while repetition structure is similar to the while structure
  ‣ Condition for repetition tested after the body of the loop is executed
  • do {
    statement
  } while ( condition );
Functions

- A Definition: A function is a named, independent section of C code that performs a specific task and optionally returns a value to the calling program or/and receives values(s) from the calling program

- There are two types of function
  - Predefined functions
    - Standard libraries like stdio.h, math.h
  - User-defined functions
    - Programmer created functions for specialized tasks
    - e.g., int fibonacci(int n)
Functions

- Characteristics of a function
- Function header: It has a return type, a unique name, and list of parameters with their types

```
Return_type function_name (type1 parameter1, type2 parameter2 ...)
{
    variable declaration(s)
    statement(s)
}
```

Examples

- `void function1 (int x, float y, char z)`
- `float function2 (float x, double y)`
- `int function3 (long size)`
- `void function4 (void)`
Functions

• The rules govern the use of variables in functions:
  
  ▶ To use a variable in a function, it must be declared either in the function header or the function body.
  
  ▶ For a function to obtain a value from the calling program (caller), the value must be passed as an argument (the actual value) unless it is a global value.

```c
/* declare and define */
int exponential (int x) {
    int result = 1;
    int i;
    for (i = 0, i < x, i++)
        result *= 2;
    return result;
}

int main() {
    /* function call */
    int y;
    y = exponential(3);
}
```
Functions

- The rules govern the use of variables in functions:
  - For a calling program (caller) to obtain a value from function, the value must be explicitly returned from the called function (callee) unless it is updated through a global variable.

```c
/* declare and define */
int exponential(int x)
{
    int result = 1;
    int i;
    for (i = 0, i < x, i++)
        result *= 2;
    return result;
}

int main()
{
    /* function call */
    int y;
    y = exponential(3);
}
```
Recursion

- Often it is difficult to express a problem explicitly
  - For example the Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...
  - It is difficult to follow the logic of this sequence

- However, a recursive definition consisting of expressing higher terms in the sequence in terms of lower terms
  - Recursive definition for \( \{f_n\} \):
    - Initialization: \( f_0 = 0, f_1 = 1 \)
    - Recursion: \( n = f_{n-1} + f_{n-2} \) for \( n > 1 \)
Recursion

- Sometimes the best way to solve a problem is by solving a smaller version of the exact same problem first.
- Recursion is a technique that solves a problem by solving a smaller problem of the same type.
- The technique ends up with functions that call themselves (recursive functions).
Logic of recursive functions

• Recursive definition and inductive proofs are complement each other

• A recursive function has two parts

• Initialization — analogous to induction base cases

• Recursion — analogous to induction step
  ‣ Recursive definition for \( \{f_n\} \):
  ‣ Initialization: \( f_0 = 0, f_1 = 1 \)
  ‣ Recursion: \( n = f_{n-1} + f_{n-2} \) for \( n > 1 \)
Recursion

• Factorial function
  ‣ Iterative implementation

```c
int Factorial(int n)
{
    int count;
    int fact = 1;
    for(count = 2; count <= n; count++)
        fact = fact * count;
    return fact;
}
```
Recursion

• Factorial function
  ‣ Recursive implementation

```c
int Factorial(int n)
{
    if (n==0)  // base case
        return 1;
    else
        return n * Factorial(n-1);
}
```
Next Class

• Stack and Heap Structures