Chapter 1
Getting Started

The University of Texas at Arlington
Dr. Ishfaq Ahmad
Topics

• Basics of Computing
• Fundamentals
• Functions
• Output and Input
• Arithmetic
• Structured Programming
Basics

- What is a program?
- What is a programming language?
- Compiling a program.
- Linking a program
- Executing a Program
What is C

• High-level Language
• Can be broken into multiple files
• Can be compiled with multiple languages
• Compilers have upgraded over time.
• Precursor to Object Oriented Programming
Identifiers

- Name of a function or variable
- Can consist of letters, digits, and underscores
- Never starts with a number
- Case sensitive
- Identifiers tend to be less than 31 characters long
- Must not be identical to any reserved word in the C Language
Identifiers

• Good Examples
  iCount
dSum
_7Days

• Bad Examples
  printf
  12months
Functions in C

• A function is needed for writing good structured code in C.
• A function is a section of code that is repeated whenever its name is called.
• Functions make C readily expandable and versatile.
Functions

• Basic Function format

    type function_name (formal parameter declaration) {
        variable declarations
        code
    }
Writing a C Program

• The beginning of any C program is the main function

```c
int main (void)
{
    return 0;
}
```
Writing a C Program

• `return` is a reserved word in C

• Ends the function and returns its current value to the calling program.
Writing a C Program

• Sample Function

```c
void does_nothing(void)
{
}
```
void does_nothing(void);

int main(void)
{
    does_nothing();
    return 0;
}

void does_nothing(void)
{
}
Printf

• The standard input/output library
• The printf statement

```c
#include <stdio.h>

int main(void)
{
    printf("This is a line of text\n");
    return 0;
}
```
Special Output Characters

\t the tab character
\b the backspace character
\” a double quote
\’ a single quote
\\ the backslash character
\0 the null character
Variables

- Stored values of a program are kept in a variable.
- Each variable has a name as of an identifier.
- Example:

  ```
  int iVar;
  ```
Variables

• Declaration of variables

```c
int iVarFirst, iVarSecond;
iVarFirst = 0;
iVarSecond = 0;
iVarFirst = 4;
iVarSecond = 6;
```
Output

• To output a value of an integer variable, the printf statement uses the format string “%d”

• Example
  printf(“Value is iVarFirst \n”);
  printf(“Value of my beloved iVarSecondis %d\n and I hate iVarFirst which is %d\n”, iVarFirst, iVarSecond);
```c
#include <stdio.h>
int main(void)
{
    int first;
    int second = 2;
    int third, fourth = 4;
    first = 1;
    third = 3;
    printf("first is %d, ", first);

    printf("second is %d, third is %d, and fourth is %d\n", third, second, fourth);

    return 0;
}
```
Input Variables - scanf

- The `scanf` command reads a value entered by the user and stores it in a variable.
- `scanf` stores the value in the memory, or the address, of the variable

```c
#include <stdio.h>
int main(void)
{
    int test, test2;
    test = 0; test2 = &test;
    printf("The address of test is %p\n", &test);
    scanf("%d", &test);
    printf("The value you entered is %d\n", test);
    scanf("%d", &test);
    printf("The value you entered is %d\n", test);
    return 0;
}
```
Arithmetic Operations

+ addition
- subtraction
* multiplication
/ division
% remainder
Arithmetic Operations

• The Remainder % function returns the remainder of a division statement

```
int w, x, y, z; x = 10; y = 3; z = 0; w = 1;
z  = -w + 4*x + y;
```
Arithmetic Operations

- Order of operations
  +,- unary operators, the sign of a variable
  *,/,% multiplicative operators of a variable
  +,- the additive operators of a variable
  = assignment operator
Arithmetic Operations

- Compound operations – parenthesis.

\[
\begin{align*}
x & = (a \times b) + c; \\
y & = (c + b) + (a \times b); \\
z & = (a + b)^2 \times (b + c) - z; \\
x & = 5; \\
x & = x + 1; \\
x & = x - 1; \\
\end{align*}
\]
Arithmetic Operations

• Increment operators
  
  \( x++; \)
  
  \( ++x; \)

• Decrement operators
  
  \( --y; \)
  
  \( y--; \)
Arithmetic Operations

• the order of the ++ or -- determines the order of operation.
• $a++*b$ is equal to $(a)*b$ then $a+1$
  
  ```
  c = a++*b;
  d = a;
  ```
• $++a*b$ is equal to $(a+1)*b$;
  
  ```
  c = ++a*b;
  d = a;
  ```
Structured Programming

• Writing programs that are
  – modular
  – modifiable
  – robust (handles errors gracefully)
  – readable
Structured Programming

- Designing a program
- Specifying what is desired of a program
- Analyzing the problem
- Designing the software solution
- Translating the design into code
- Testing and debugging the code
- Maintaining and modifying the code as needs arise.
Structured Programming

• C Code can be broken up into functions
• Each function behaves as its own sub program
• Each function can be tested independently
• Each function can be reused by the parent program.
• Stepwise Refinement – breaking up a large task into smaller tasks, and bringing them together into a single structure.
Structured Programming

• Preprocessor Constants
  – C provides the ability to declare a constant
  – The constant has the same value for all the times it appears in the code
  – uses the `#define` preprocessor statement
    • `#define MAXSECONDS 3600`
  – does not end with a semicolon
  – must recompile each time it is changed
  – useful for a value that is repeated many times but changes seldom.
Structured Programming

• Function Parameters
  – Functions can have a value passed to them to be used in their code.
  – In mathematics, a function $f$ using variable $x$ for a quadratic would be $f(x)=x^2+2x+1$;
  – In C, the function would look like
    ```c
    int Quadratic(int x)
    {
        int result=0;
        result=x*x+2*x+1;
        return result;
    }
    ```
Structured Programming

• Function Calls

And would be called by the main function:

```c
int main(void)
{
    int value;
    value = Quadratic(5);
    printf("f(5)=%d\n",value);
    value = Quadratic(3);
    printf("f(3)=%d\n",value);
}
```

And the resulting output would look like:

f(5)=36
f(3)=16