Introduction to C Programming

CEN111 Algorithms and Programming I

A Simple C Program

```
1. // A first program in C.
2. #include <stdio.h>
3.
4. // function main begins program execution
5. int main(void) {
6.    printf("Welcome to C!\n");
7.    return 0;
8. } // end function main
```

- **OUTPUT:**
  - `Welcome to C!`

A Simple C Program

Comments

- Begin with //
- Insert comments to document programs and improve program readability
- Comments do not cause the computer to perform actions
- Help other people read and understand your program
- Can also use /* … */ multi-line comments
  - Everything between /* and */ is a comment

A Simple C Program

`#include` Preprocessor Directive

- `#include <stdio.h>`
- C preprocessor directive
- Preprocessor handles lines beginning with # before compilation
- This directive includes the contents of the standard input/output header (<stdio.h>)
  - Contains information the compiler uses to ensure that you correctly use standard input/output library functions such as `printf`

A Simple C Program

Blank Lines and White Space

- Blank lines, space characters and tab characters make programs easier to read
- Together, these are known as **white space**
- Generally ignored by the compiler

A Simple C Program

The `main` Function

- Begins execution of every C program
- Parentheses after `main` indicate that `main` is a **function**
- C programs consist of functions, one of which must be `main`
- Precede every function by a comment stating its purpose
- Functions can return information
- The keyword int to the left of main indicates that main “returns” an integer value

A Simple C Program

The `main` Function

- Functions can receive information
  - `void` in parentheses means `main` does not receive any information
- A left brace, `{`, begins each function's **body**
- A corresponding **right brace**, `}`, ends each function's body
- A program terminates upon reaching `main`'s closing right brace
- The braces form a block

A Simple C Program

An Output Statement

- `printf("Welcome to C!\n");`
- Performs an **action**—displays the **string** in the quotes
  - A string is also called a **character string**, a **message** or a **literal**
- The entire line is called a **statement**
- Every statement ends with a semicolon **statement terminator**
- Characters usually print as they appear between
  - Notice the characters `\n` were not displayed.

A Simple C Program

Escape Sequences

- In a string, backslash (`\`) is an **escape character**
- Compiler combines a backslash with the next character to form an **escape sequence**
- **`\n`** means **newline**
- When `printf` encounters a newline in a string, it positions the output cursor to the beginning of the next line

A Simple C Program

Escape Sequence	Description
\n	Moves the cursor to the beginning of the next line.
\t	Moves the cursor to the next horizontal tab stop.
\a	Produces a sound or visible alert without changing the current cursor position.
****	Because the backslash has special meaning in a string, **** is required to insert a backslash character in a string.
\”	Because strings are enclosed in double quotes, **\”** is required to insert a double-quote character in a string.

A Simple C Program

The Linker and Executables

- When compiling a `printf` statement, the compiler merely provides space in the object program for a “call” to the function
- The compiler does not know where the library functions are—the linker does
- When the linker runs, it locates the library functions and inserts the proper calls to these functions in the object program
- Now the object program is complete and ready to execute
- The linked program is called an **executable**

A Simple C Program

The return Statement

- Last line in the main function.
- Transfers control from your program to the operating system.
- The value 0 ,in this case, indicates that your program executed without an error.
- Right brace } indicates end of main has been reached

A Simple C Program

Indentation Conventions

- Indent the entire body of each function one level of indentation within the braces that define the function's body
- Emphasizes a program's functional structure and helps make them easier to read
- Set an indentation convention and uniformly apply that convention
- Style guides often recommend using spaces rather than tabs

A Simple C Program

```
1. // Printing on one line with two printf statements.
2. #include <stdio.h>
3.
4. // function main begins program execution
5. int main(void) {
6.    printf("Welcome ");
7.    printf("to C!\n");
8.    return 0;
9. } // end function main
```

- **OUTPUT:**
  - `Welcome to C!`

A Simple C Program

Using Multiple `printfs`

- Next program uses two statements to produce the same output as previous one.
- Works because each `printf` resumes printing where the previous one finished
- Line 6 displays `Welcome` followed by a space (but no newline)
- Line 7's `printf` begins printing immediately following the space

A Simple C Program

Displaying Multiple Lines with a Single `printf`

- One `printf` can display several lines
- Each `\n` moves the output cursor to the beginning of the next line

A Simple C Program

```
1. // Printing multiple lines with a single printf.
2. #include <stdio.h>
3.
4. // function main begins program execution
5. int main(void) {
6.    printf("Welcome\nto\nC!\n");
7.    return 0;
8. } // end function main
```

- **OUTPUT:**

```
Welcome
to
C!
```

A Simple C Program

Displaying Multiple Lines with a Single `printf`

- One `printf` can display several lines
- Each `\n` moves the output cursor to the beginning of the next line

Adding Two Integers

- `scanf` standard library function obtains information from the user at the keyboard
- Next program obtains two integers, then computes their sum and displays the result

Adding Two Integers

```
1.// Addition program.
2.#include <stdio.h>

3.// function main begins program execution
4.int main(void) {
5.    int integer1 = 0; // will hold first number
6.    int integer2 = 0; // will hold second number
7.    printf("Enter first integer: "); // prompt
8.    scanf("%d", &integer1); // read an integer
```

Adding Two Integers

```
9.     printf("Enter second integer: "); // prompt
10.    scanf("%d", &integer2); // read an integer
11.    int sum = 0; // variable in which sum will
  be stored
12.    sum = integer1 + integer2; // assign total
  to sum
13.    printf("Sum is %d\n", sum); // print sum
14.    return 0;
15.} // end function main
```

Adding Two Integers

- OUTPUT

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

Adding Two Integers

Variables and Variable Definitions

- Lines 5 and 6 are definitions.
- The names `integer1` and `integer2` are variables—locations in memory where the program can store values for later use
- `integer1` and `integer2` have type `int`—they'll hold whole-number integer values
- Lines 5 and 6 initialize each variable to 0

Adding Two Integers

Define Variables Before They Are Used

- All variables must be defined with a name and a type before they can be used in a program
- You can place each variable definition anywhere in main before that variable's first use in the code
- In general, you should define variables close to their first use

Adding Two Integers

Identifiers and Case Sensitivity

- A variable name can be any valid identifier
- Each identifier may consist of letters, digits and underscores (_), but may not begin with a digit
- C is case sensitive, so `a1` and `A1` are different identifiers
- A variable name should start with a lowercase letter
- Choosing meaningful variable names helps make a program self-documenting
- Multiple-word variable names can make programs more readable
  - separate the words with underscores, as in total_commissions, or
  - run the words together and begin each subsequent word with a capital letter as in totalCommissions.

Adding Two Integers

Prompting Messages

- Line 7 displays "Enter first integer: "
- This message is called a prompt
  - tells user to take a specific action

Adding Two Integers

The `scanf` Function and Formatted Inputs

- Line 8 uses `scanf` to obtain a value from the user
- Reads from the standard input, usually the keyboard
- The "`%d`" is the format control string — indicates the type of data the user should enter (an integer)
- Second argument begins with an ampersand (&) followed by the variable name
  - Tells scanf the location (or address) in memory of the variable
  - scanf stores the value the user enters at that memory location

Adding Two Integers

Prompting for and Inputting the Second Integer

- Line 9 prompts the user to enter the second integer
- Line 10 obtains a value for variable `integer2` from the user.

Adding Two Integers

Defining the `sum` Variable

- Line 11 defines the int variable `sum` and initializes it to 0 before we use `sum` in line 12.

Adding Two Integers

Assignment Statement

- The assignment statement in line 12 calculates the total of `integer1` and `integer2`, then assigns the result to variable `sum` using the assignment operator (=)
- Read as, “sum gets the value of the expression integer1 + integer2.”
- Most calculations are performed in assignments

Adding Two Integers

Binary Operators

- The = operator and the + operator are binary operators—each has two operands
- Place spaces on either side of a binary operator to make the operator stand out and make the program more readable

Adding Two Integers

Printing with a Format Control String

- The format control string "Sum is %d\n" in line 13 contains some literal characters to display ("Sum is ") and the conversion specification `%d`, which is a placeholder for an integer
- The sum is the value to insert in place of `%d`

Adding Two Integers

Combining a Variable Definition and Assignment Statement

- You can initialize a variable in its definition
- For example, lines 11 and 12 can add the variables `integer1` and `integer2`, then initialize the variable sum with the result:
  - `int sum = integer1 + integer2; // assign total to sum`

Adding Two Integers

Calculations in `printf` Statements

- Actually, we do not need the variable sum, because we can perform the calculation in the `printf` statement
- Lines 11–13 can be replaced with
  - `printf("Sum is %d\n", integer1 + integer2);`

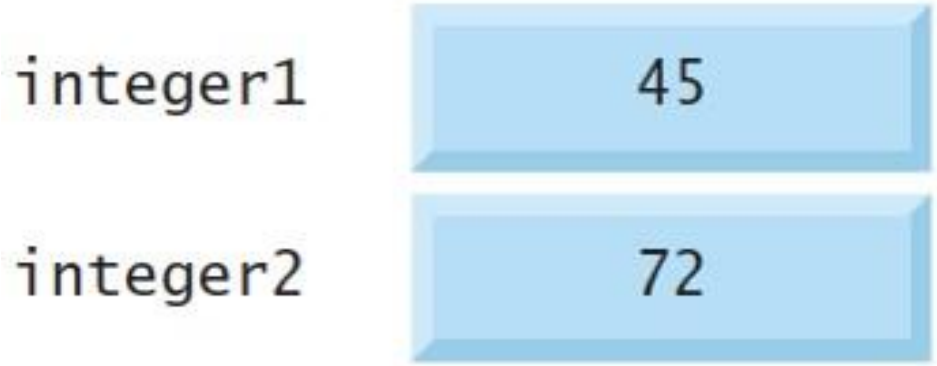
Memory Concepts

- Every variable has a name, a type, a value and a location in the computer's memory
- After placing 45 in `integer1`

- When a value is placed in a memory location, it replaces the location's previous value which is lost

Memory Concepts

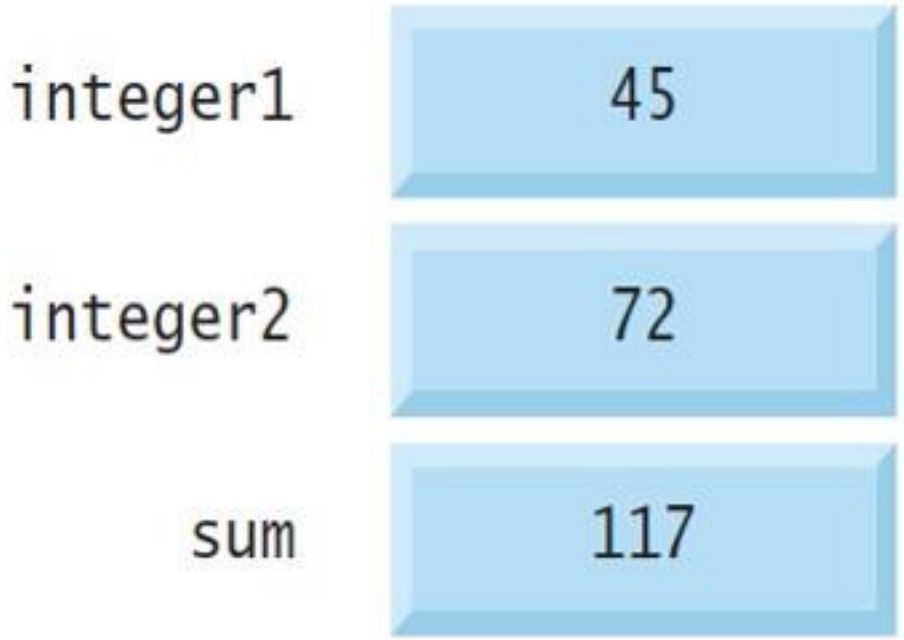
- After placing 72 in `integer2`



- These locations are not necessarily adjacent in memory

Memory Concepts

- After calculating sum of `integer1` and `integer2`



Language Elements

- Token: the smallest element of a program that is meaningful to the compiler.
- Kinds of tokens in C:
  - Reserved words
  - Identifiers
  - Constants
  - Operators
  - Punctuators

Reserved Words

- A word that has a special meaning in C
- Identifiers from standard library and names for memory cells
- Appear in lowercase
- Cannot be used for other purposes

Reserved Words

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Identifiers

- Identifiers are used for:
  - Variable names
  - Function names
  - Macro names

Identifiers

- An identifier must consist only of letters, digits, and underscores.
- An identifier cannot begin with a digit.
- A C reserved word cannot be used as an identifier.
- An identifier defined in a C standard library should not be redefined.
- Case-sensitive e.g. `Ali` and `ali` are two different identifiers.

Identifiers

- Valid Identifiers

  letter_1, letter_2, inches, CENT_PER_INCH,
  Hello, variable

- Invalid Identifiers

  1Letter double int TWO*FOUR joe’s

Variable Declarations

- Variable
  - a name associated with a memory cell whose value can change
- Variable declarations
  - statements that communicate to the compiler the names of variables in the program and the kind of information stored in each variable
  - Variables must be declared before use, a syntax (compile-time) error if these are violated
- Every variable has a name, a type, a size and a value

Basic Datatypes in C

- Integer — int
- Character — char
- Floating Point — float
- Double precision floating point — double

- Datatype modifiers
  - signed / unsigned (for int and char)
  - short / long

Basic Datatypes in C

• signed char (8 bits)	-127 to +127
• unsigned char	0 to 255
• short int (16 bits)	-32,767 to +32,767
• unsigned short int	0 to 65,535
• int (32 bits)	-2,147,483,647 to +2,147,483,647
• unsigned int	0 to 4,294,967,295
• long int (32-64 bits)	-2,147,483,647 to +2,147,483,647
• unsigned long int	0 to 4,294,967,295
• float	~10^-37 to ~10^38
• double	~10-^307 to ~10^308
• long double	~10^-4931 to ~10^4932

Basic Datatypes in C

- Placeholders in format string
  - `%c`: the argument is taken to be a single character
  - `%d`: the argument is taken to be an integer
  - `%f`: the argument is taken to be a floating point (float or double)
  - `%s`: the argument is taken to be a string

Variable Declarations

- A declaration consists of a data type name followed by a list of (one or more) variables of that type:
  - `char c;`
  - `int num1, num2;`
  - `float rate;`
  - `double avarage;`

Variable Declarations

- A variable may be initialized in its declaration.
  - `char c = 'a';`
  - `int a = 220, b = 448;`
  - `float x = 0.00123;`
  - `double y = 123.00;`

Variable Declarations

- Variables that are not initialized may have garbage values.
- Whenever a new value is placed into a variable, it replaces the previous value
- Reading variables from memory does not change them

Operators

- Arithmetic operators
- Assignment operator
- Logical operators

Arithmetic in C

Arithmetic Operator	Meaning	Example
+	addition	5 + 2 is 7 5.0 + 2.0 is 7.0
–	subtraction	5 – 2 is 3 5.0 – 2.0 is 3.0
*	multiplication	5 * 2 is 10 5.0 * 2.0 is 10.0
/	division	5.0 / 2.0 is 2.5 5 / 2 is 2
%	remainder	5 % 2 is 1

Arithmetic in C

- Integer Division and the Remainder Operator
  - Integer division yields an integer result
  - 7 / 4 evaluates to 1
  - 17 / 5 evaluates to 3
- Integer-only remainder operator, %, yields the remainder after integer division
  - 7 % 4 yields 3
  - 17 %5 yields 2
- An attempt to divide by zero usually is undefined
  - Generally, a fatal error
  - Nonfatal errors allow programs to run to completion, often with incorrect results

Arithmetic in C

Parentheses for Grouping Subexpressions

- Parentheses are used in C expressions in the same manner as in algebraic expressions

Rules of Operator Precedence

- Generally the same as in algebra
  - Expressions grouped in parentheses evaluate first
    - In nested parentheses, operators in the innermost pair of parentheses are applied first
  - *, / and % are applied next left-to-right
  - + and - are evaluated next left-to-right
  - The assignment operator (=) is evaluated last.

Arithmetic in C

$$m = \frac{a + b + c + d + e}{5}$$

C: $\mathrm{m} = (\mathrm{a} + \mathrm{b} + \mathrm{c} + \mathrm{d} + \mathrm{e})/5;$

- Parentheses are required above

Arithmetic in C

$y = mx + b$

C: $y = m * x + b;$

Arithmetic in C

$z = pr \bmod q + w/x - y$

C:

```
z = p * r % q + w / x - y;
```

Step 1. `y = 2 * 5 * 5 + 3 * 5 + 7;` *(Leftmost multiplication)*

`2 * 5 is 10`

Step 2. `y = 10 * 5 + 3 * 5 + 7;` *(Leftmost multiplication)*

`10 * 5 is 50`

Step 3. `y = 50 + 3 * 5 + 7;` *(Multiplication before addition)*

`3 * 5 is 15`

Step 4. `y = 50 + 15 + 7;` *(Leftmost addition)*

`50 + 15 is 65`

Step 5. `y = 65 + 7;` *(Last addition)*

`65 + 7 is 72`

Step 6. `y = 72` *(Last operation—place 72 in y)*

Arithmetic in C

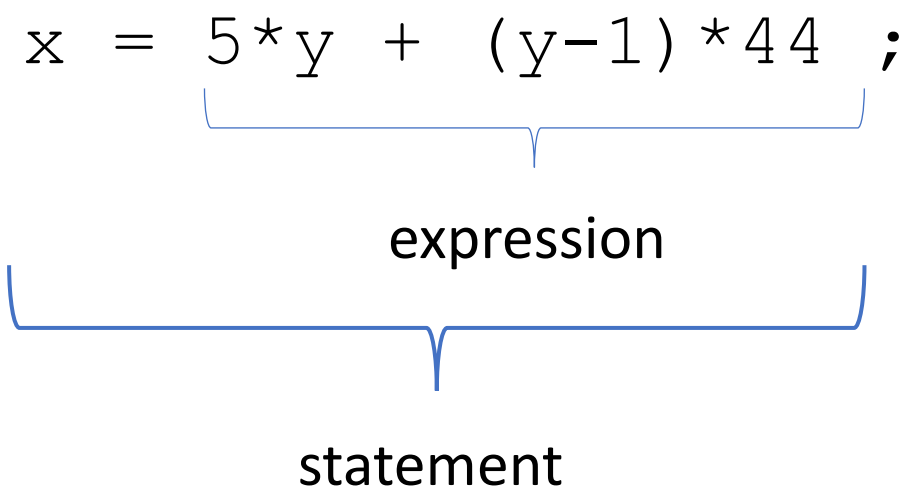
Using Parentheses for Clarity

- Redundant parentheses can make an expression clearer

```
y = (a * x * x) + (b * x) + c;
```

Assignment Operator

- variable = expression;
- expressions:
  - operations
  - variables
  - constants
  - function calls



- Precedence of the assignment operator is lower than the arithmetic operators

Assignment Operator

- an instruction that stores a value of a computational result in a variable

- `x=y ;` l-value vs. r-value
- `x+1 = 3;` invalid l-value expression

- l-value usages must refer to a fixed position in the memory

Increment and Decrement Operators

- **Postfix Increment/Decreement**

```
int a=3;
a++;      \\ a=a+1
```

- The value of the postfix increment expression is determined before the variable is increased

```
x=a++;
```

1. `x=a;`
2. `a=a+1;`

Increment and Decrement Operators

- **Prefix Increment/Decreement**

```
int a=3;
++a;      \\ a=a+1
```

- The effect takes place before the expression that contains the operand is evaluated

```
x=++a;
```

1. `a=a+1;`
2. `x=a;`

Increment and Decrement Operators

```
/*** increment and decrement expressions ***/
#include <stdio.h>
int main(void) {
   int a =0 , b = 0, c = 0;
   a = ++b + ++c;
   a = b++ + c++;
   a = ++b + c++;
   a = b-- + --c;
   return 0;
}
```

Compound Assignment Operator

```
sum=sum+x;
```

- can be abbreviated
  - `sum+=x;`

- operand1=operand1 operator operand2
- operand1 operator=operand2

Integer/Float Conversions

- An arithmetic operation between an integer and an integer always yields an integer result .
- An arithmetic operation between a float and a float always yields a float result.
- In an arithmetic operation between an integer and a float, the integer is first promoted to float and then the operation is carried out. Hence, it always yields a float result.
- On assigning a float to an integer, the float is demoted to an integer
- On assigning an integer to a float, it is promoted to a float.

Homework

- Write a C program to reverse a given 5 digits number.