My First C Program
Variables & Constants

1. Objective:

- To understand the structure of a C-language program.
- To write your first C program.
- To introduce the Include preprocessor command.
- To be able to create good identifiers for objects in a program.
- To be able to list, describe, and use the C basic data types.
- To be able to create and use variables and constants.
- To understand input and output concepts.
- To be able to use simple input and output statements.

2. What is C?

- The C programming language was designed by Dennis Ritchie at Bell Laboratories in the early 1970s (Headquarters in Murray Hill, New Jersey)
- C is a structured programming language.
- It is considered a high-level language because it allows the programmer to concentrate on the

problem at hand and not worry about the machine that the program will be using.

- An December 2015 survey ranking language by their usage (lines of code written) yielded the following:

Language	Usage by percentage
Java	21%
C	17%
C++	6%
Python	5%
C#	4%
PHP	3%
Visual Basic .NET	2%
Javascript	2%
Perl	2.2%
Ruby	2%
Assembly language	1%
(Source: http://www.tiobe.com)	

3. Structure of a C program:

/* Comments: this is my first C program */
Preprocessor Directives
Global Declarations

// This is the main function
Int main(**void**) {
Local Declarations
Statements
}

//Other functions.

- My First C program

```
//A first program in C
#include  <stdio.h>
void  main(void)
{
   printf("Welcome to C!\n");
}
```

- **Comments & Whitespace**
    - Used to describe program
    - Text surrounded by comments symbols is ignored by computer
    - Two types of comments:
        - Single-line comment uses the **//** symbols
        - Multi-line comment uses the **/*** and ***/** symbols
    - Whitespace
        - It is used to make the program more readable.
        - It refers to:
            - blank spaces between items within a statement, and
            - blank lines between statements.
        - A compiler ignores most whitespace.

- **Preprocessor Directives**
    - Tells computer to load contents of a certain file
    - Example:

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

    - stdio.h:

    - Allows standard input/output operations file and console (also a file) Input-Output: scanf, printf, open, close, read, write, perror, etc.

- stdlib.h:
    - Common utility functions: malloc, calloc, strtol, atoi, etc

- string.h:
    - String and byte manipulation: strlen, strcpy, strcat, memcpy, memset, etc.

- ctype.h:
    - Character types: isalnum, isprint, isupport, tolower, etc.

- errno.h:
    - Defines errno used for reporting system errors

- math.h:
    - Math functions: ceil, exp, floor, sqrt, etc.
    - Example: In class

```
//gcc 5.4.0
// By A. Bellaachia
// Computer the square root of a number
#include  <stdio.h>
#include <math.h>
int main(void)
{
   int value1;
   int value2;
   int sum;

   value2 = 100;
   value1 = 16;

   sum = sqrt(value1) + sqrt(value2);
   printf("The sume is: %d\n", sum);

   return 0;
}
```

- time.h:
  - Time related facility: asctime, clock, time_t, etc.

- **Global Declarations:**
    - A set of declarations that are used by your program.
    - They can be variables or functions.

- **int main():**
    - C programs contain one or more functions, exactly one of which must be **main**
    - Parenthesis used to indicate a function
    - **int** means that **main** "returns" an integer value
    - Braces (**{** and **}**) indicate a block
    - The bodies of all functions must be contained in braces
    - printf("Welcome to C!\n"):
        - Instructs computer to perform an action
        - Specifically, prints the string of characters within quotes (**" "**)
        - Entire line called a statement
        - All statements must end with a semicolon (**;**)
        - Escape character (\):
            - Indicates that printf should do something out of the ordinary **\n** is the newline character

- **Example**: From "Programming in C", zyBooks

Which statement prints: Welcom	printf(Welcome!);
	printf "Welcome!";
	printf("Welcome!");

Which statement prints Hey followed by a new line?	printf(Hey\n);
	printf("Hey"\n);
	printf("Hey\n");

- **return 0;**
  - A way to exit a function in this case, means that the program terminated normally

- Another C program:
  - Add two numbers: In class

```
#include  <stdio.h>
int main(void)
{
   int value1;
   int value2;
   int sum;
```

```
    value2 = 100;
    value1 = 10;

    sum = value1 + value2;
    printf("The sume is: %d\n", sum);

    return 0;
}
```

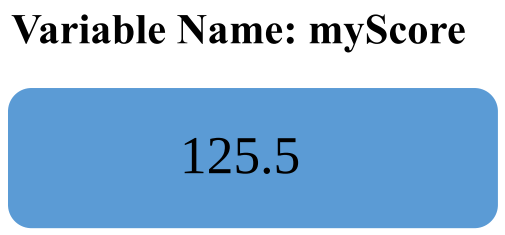
4. Declarations

- To notify the compiler about our needs in term of memory cells
- Used to create both variables and constants
- A declaration gives the basic underlying type of the variable and optionally its initial value.
- An unbroken rule of C, never broken:
    - C requires that anything you use must have been previously defined: variables as well as constants, procedures, functions, and all other entities.
    - C never breaks this rule.

5. Variables & Assignments

- A variable represents a memory location used to store data for your program.

- Each variable is defined by the following attributes:



  - **A memory** cell used to hold its value
  - **A unique identifier or name** (a name given by the programmer)
  - **A data type** (what type is your variable? a number, a word, etc.)

- The programmer must **define a variable before any statement that assigns or reads the variable**, so that the variable's memory location is known.

- To ask the compiler to reserve a memory location for your data, you need a declaration in C.

- Here is a declaration statement:

```
int value1;
```

- The compiler allocates a memory location for value1 capable of storing an integer, hence the "int".

6. Assignments

- It is a statement (like a sentence in natural language):

variableName = expression;

- An expression may be:
  - A number like 80,
  - A variable name,
  - Or a simple calculation

- Examples:
  - value1 = 200;
  - value1 = value2;
  - value1 = value1+1;
  - Etc.

- Multiple declarations: In class

```
//gcc 5.4.0

#include  <stdio.h>


// Print an multiple decalarations.
int main(void)
{
   int x, y, z;

   x = 0; y = 10; z = 1000;

   printf(" x: %d\n", x);
   printf(" y: %d\n", y);
   printf(" z: %d\n", z);
}
```

- The value of ***a variable*** may be changed during the execution of a program:
  - value1 = 10;
  - value1 = sqrt(400);

- Variable Initialization:
  - A good practice is to initialize a variable before using it.
  - Example: In class

```
//gcc 5.4.0

#include  <stdio.h>


// Print an uninitialized varaible.
int main(void)
{
   int myScore;

   //Print myScore before it is initialized. Some compilers
may print garbage.
   printf("My Score is : %d\n", myScore);

   //Initial myScore now
   myScore = 105;
   printf("My Score is : %d\n", myScore);


   return 0;
}
```

7. Questions/Practice

1. int dogCount;
    - A. Error
    - B. No error

2. int amountOwed = -999;

    - A. Error
    - B. No error

3. int numYears = 9000111000;

    - A. Error
    - B. No error

4. Define an integer variable named numPeople. Do not initialize the variable.

    ____

5. Define an integer variable named numDogs, initializing the variable to 0 in the definition.

____

6. Define an integer variable named daysCount, initializing the variable to 365 in the definition.

____

7. Write an assignment statement to assign 99 to numCars.

____

8. Assign 2300 to houseSize.

____

9. Assign the current value of numApples to numFruit.

____

10. The current value in houseRats is 200. Then:

```
numRodents = houseRats;
```

executes. You know 200 will be stored in numRodents. What is the value of houseRats after the statement executes? Valid answers: 0, 199, 200, or unknown.

____

11. Assign the result of ballCount - 3 to numItems.

____

12. dogCount is 5. After

```
animalsTotal = dogCount - 3;
```

executes, what is the value in animalsTotal?

____

13. dogCount is 5. After

```
animalsTotal = dogCount - 3;
```

executes, what is the value in dogCount?

____

14. What is the value of numBooks after both statements execute?

```
numBooks = 5;
numBooks = 3;
```

____

15. numApples is initially 5. What is numApples after:

```
numApples = numApples + 3;
```

____

16. numApples is initially 5. What is numFruit after:

```
numFruit  = numApples;
numFruit  = numFruit + 1;
```

____

17. Write a statement ending with - 1 that decreases variable flyCount's value by 1.

____

18. What is the value of each assignment in the following code:

```
w = 1;
y = 2;
z = 4;

x = y + 1;
w = 2 - x;
z = w * y;
```

19. What is the value of each assignment in the following code:

```
x = 4;
y = 0;
z = 3;

x = x - 3;
y = y + x;
z = z * y;
```

20. What is the value of each assignment in the following code:

```
x = 6;
y = -2;
```

```
y = x + x;
w = y * x;
z = w - y;
```

21. What is the value of each assignment in the following code:

```
w = -2;
x = -7;
y = -8;

z = x - y;
z = z * w;
z = z / w;
```

8. Identifiers

- An identifier is the name a programmer gives to a variable or a function.
- The characters used to create an identifier are:
    - Letters: a-z and A-Z,
    - Digits: 0-9
    - Underscore character: "_"
- **The name of an identifier must start with a letter.**

- **C is a case-sensitive language**

- **Examples**:
    - Valid identifiers:
        - c, cat, Cat, n1m1, short1, and _hello.
        - Note that cat and Cat are different identifiers.
    - Invalid identifiers:
        - 42c (starts with a digit)
        - hi there (has a disallowed symbol: space),
        - cat! (has a disallowed symbol: !).

- **Reserved Names (keywords)**:
    - A reserved word is a word that is part of the language, like int, short, or double.

- A programmer cannot use a reserved word as an identifier.
- A list of reserved words appears at the end of this section. (https://www.programiz.com)

auto	else	Long	Switch
break	enum	Register	Typedef
case	extern	Return	Union
char	float	Short	Unsigned
const	for	Signed	Void
continue	goto	Sizeof	Volatile
default	if	Static	While
do	int	Struct	_Packed
double			