Week 4

Hari Sethuraman

By the end of today

You should have a solid understanding of:

- Operators
- If-Conditions
- Else

Recap of Last lesson

- Datatypes: allow you to store different types of information
  - Int: Integer (4 bytes)
  - Char: Character (1 byte)
  - Float: number with decimals (4 bytes)
  - Boolean: true of false (1 byte)

How a C-program looks like

```
#include <stdio.h>
int main ()
{

}
```

Header

Main function

Insert code here
(between curly brackets)

Header and main function

- The Header (The <stdio.h>) imports a lot of useful features and functions into the program.
  - There are other headers that bring other functions, but <stdio.h> will be in every program
- The main function is a function that is called by default when the program starts
  - All the logical code you write must be inside a function
  - The main function always looks like: *int main() {…}*

Recap (continuation)

- Printf: prints out whatever you pass to it:
  - Printf(“Hi”);
    - Prints Hi
  - Include ‘\n’ for a new line (like pressing enter)
  - Placeholders allow you to insert variables in a print statement
    - %c for characters
    - %d or %i for integers
    - %f for floats
    - Printf(“Hello, %c\n”, c);
      - Will print ‘Hello’ followed by the value of *c*.

Recap (continuation)

- To get the input from the user and store it in a variable, we use *scanf*
  - Scanf consists of two parts
    - Scanf(“ ”,);

&variable name

Placeholder with matching data-type

  - Example: get an integer input and store it in the variable *I*
    - *int i = 0;* **initializing the variable value**
    - *scanf(“%d”, &i);* **get input from the user**
  - The computer stores the input once the user pressers the enter button
  - *Why the ‘&’? We use this symbol in order to tell the computer the location in memory of *i*. if we just used *i*, we would pass the value of *i*, which is currently *0*.
  - This is not very useful, therefore, by using the ‘&’ before the variable name, we pass the location of the variable in memory rather than the value, so that the computer can directly write the input value there

Operators

- For any kind of logic or operation, we use ‘Operators’. There are 3 main types:
  - Arithmetic operators
  - Relational operators
  - Logical operators

Arithmetic operators

- Used to computer arithmetic operations like addition, subtraction…
  - + used to add numbers
    - Example
      - Int x = 3 + 5 + 7; (x = 15)
  - - used to subtract numbers
  - * used to multiply numbers
  - / for dividing
  - % is the ‘modulo operator’. Returns the remainder of division:
    - Int x = 5;
    - Int y = 2;
    - Int z = x % y; Remainder of dividing first number by second
      - Z = 1

Relational operators

Int a = 1;
Int b = 0;

- Used to compare variables or numbers (returns a true or false value)
  - == used to check if two numbers or variables are equivalent
    - Not assignment! (=)
    - (a == b) -> returns false
  - != used to check if two numbers or variables are not equal
    - (a != b) -> returns true
  - > and < used to check if one variable or number is greater or lesser than the other
    - (a > b) -> returns true
  - <= and >= used for ‘lesser than or equal to’ and ‘greater than or equal to’

Logical operators

```
bool a = true;
bool b = false;
```

To use bool, import <bool.h>

- Compares two Boolean values and returns a Boolean value
  - &&: means ‘and’
    - Both values must be true to return true
    - a && b -> returns false because b is false
  - ||: means ‘or’
    - At least one operation must be true to return true
    - a || b -> returns true because a is true
    - True || true -> true
    - True || false -> true
    - False || true -> true
    - False || false -> false

Question

- int a = 2
- Int b = 5
- Int c = b + a
- Int d = (b*2) % (a*2)
- Question: (c != b) || (d == a)
  - What does this return?

Answer

- a = 2
- b = 5
- c = a + b ->7
- d = (2*b) % (2*a) -> 2
- (c != b) || (d == a)
- true || true
- true

If conditions

- If conditions run the code inside them when a certain condition is met:

```
if (          )
{

}
```

Condition goes between these brackets

Enter code between curly braces

If the condition inside brackets is true, then run code between curly braces

Example

- Get two integers from the User
  - Print ‘a equals b’ if they are equal

Program

```
# include <stdio.h>

int main ()
{
        int a = 0;
        int b = 0;
        scanf(“%d”, &a);
        scanf(“%d”, &b);

        if (a == b)
        {
                printf(“%d equals %d\n”, a, b);
        }
}
```

Else

- We use ‘else’ after an if condition:
  - If __ is true, then execute __. Else, execute __
  - Does not require a condition

```
else
{

}
```

Enter code between curly braces

Example

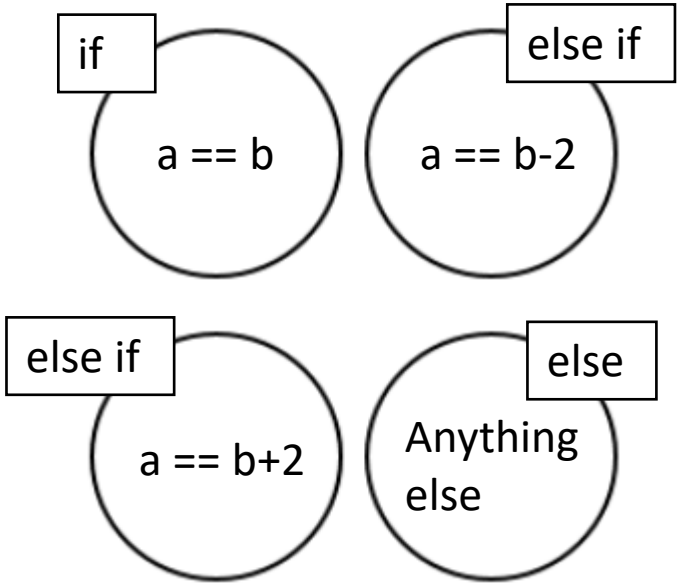
- Get two integers from the User
  - Print ‘a equals b’ if they are equal
  - Print ‘a does not equal b’ if they are not equal

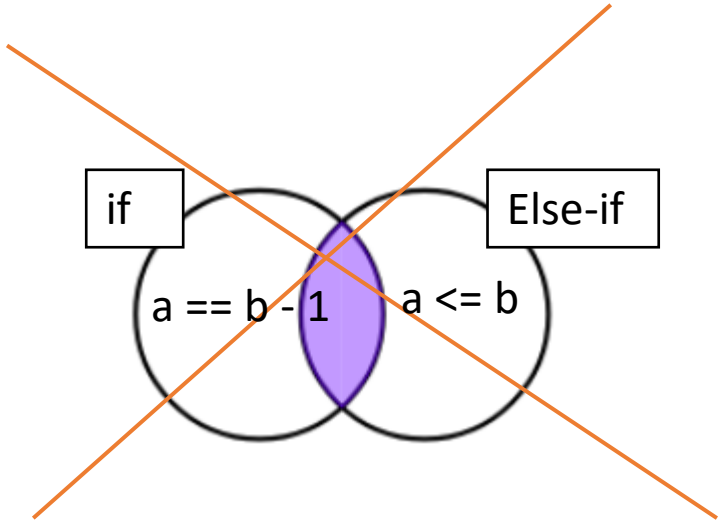
Program

```
# include <stdio.h>

int main ()
{
        int a = 0;
        int b = 0;
        scanf("%d", &a);
        scanf("%d", &b);

        if (a == b)
        {
                printf("%d equals %d\n", a, b);
        }
        else
        {
                printf("%d does not equal %d\n", a, b);
        }
}
```

Else if

- Use else if statements if you want to stack multiple ifs, and only want one of them to execute
  - Always comes after an if-condition
  - IMPORTANT: all ifs, else ifs, and else's must have Mutually Exclusive conditions and code





Example

- Get two integers from the User
  - Print ‘a equals b’ if they are equal
  - Print ‘a is greater than b by 1’ if a is 1 greater than b
  - Print ‘a is lesser than b by 1’ if a is 1 lesser than b
  - Otherwise, print ‘hello world’

Program

Simple Calculator Program

- Get input from user as a character *c*
- Get input for two numbers
- If *c* is:
  - ‘a’: add the two numbers
  - ‘s’: subtract the numbers
  - ‘d’: divide
  - ‘m’: multiply
  - ‘r’: modulo
  - Something else, print ‘error’