

Recall Format specification for printf and scanf

- Format specifier is a % followed by a character
- To print a literal % use %%
- Depending on what follows % we have the following meanings
 - d, x, o integer in decimal, hex or octal
 - f float
 - g generic floating point, i.e. float or double.
 - c character
- For more formatting characters check out man page of scanf.

```
#include <stdio.h>
int main()
{
    double h,w;
    double area,per;
    printf("Enter the height and width: ");
    scanf("%g %g",&h,&w);
    area = h * w;
    per = 2 * (h + w);
    printf("\nFor a %g x %g rectangle\n\t",h,w);
    printf("Area = %g\n\tPerimeter = %g\n", area, per);
}
```

OUTPUT

```
Enter the height and width: 2.5 10.0
For a 2.5 x 10.0 rectangle
    Area = 25
    Perimeter = 25
```

Operators in C

- **Binary operators:** +, -, *, /, %
e.g.: i+j, x*i-j, y%z
- **Unary Operators:** +, -
e.g.: i++, --j
- **Relational Operators:** <, ==, >, <=, >=, !=
e.g.: ii < jj, k >= l

Operators in C

- **Logical Operators:** &&, ||
e.g.: i < lim-1 && (c=getchar())
- **Bitwise Operators:** &, |, ^, >>, <<, ~
These six bitwise operators may only be applied to integral operands, that is, char, short, int, and long, whether signed or unsigned.

e.g. The bitwise OR operator | is used to turn bits on:
x = x | SET_ON;
sets to one in x the bits that are set to one in SET_ON

Expressions combine variables and constants to produce new values.

1)

```
int i=4, n=2, s=5 ;
n = 10 * n + (s*i - '0');
```

2)

```
unsigned long int next = 1;
next = next * 1103515245 + 12345;
```

3)

```
int x=10, n=1;
x = n++;
```

4)

```
int x=10, n=1;
x = ++n;
```

5)

```
Boolean j;
char s, t='a';
j = ((s = t) != '\0');
```

More Comments on Logical Operators

- **&&** (logical AND)
 - Returns **true** if both conditions are **true**
- **||** (logical OR)
 - Returns **true** if either of its conditions are **true**
- **!** (logical NOT, logical negation)
 - Reverses the truth/falsity of its condition
 - Unary operator, has one operand
- Useful as conditions in loops

<u>Expression</u>	<u>Result</u>
true && false	false
true false	true
!false	true

Confusing Equality (==) and Assignment (=) Operators

- Dangerous error
 - Does not ordinarily cause syntax errors
 - Any expression that produces a value can be used in control structures
 - Nonzero values are **true**, zero values are **false**
 - Example using ==:


```
if ( payCode == 4 )
    printf( "You get a bonus!\n" );
```

 - Checks **paycode**, if it is **4** then a bonus is awarded

Confusing Equality (==) and Assignment (=) Operators

- Example, replacing == with =:


```
if ( payCode = 4 )
    printf( "You get a bonus!\n" );
```

 - This sets **paycode** to **4**
 - **4** is nonzero, so expression is **true**, and bonus awarded no matter what the **paycode** was
- Logic error, not a syntax error

Confusing Equality (==) and Assignment (=) Operators

- lvalues
 - Expressions that can appear on the left side of an equation
 - Their values can be changed, such as variable names
 - `x = 4;`
- rvalues
 - Expressions that can only appear on the right side of an equation
 - Constants, such as numbers
 - Cannot write `4 = x;`
 - Must write `x = 4;`
 - lvalues can be used as rvalues, but not vice versa
 - `y = x;`

•C does automatic conversion between integers and floats

- Integer to Float/Double extension
- Float/Double to integer truncation
- Unfortunately this is a very bad design.

```
#include<stdio.h>
main()
{
    int u = 10;
    int v = 11;
    float av;
    av = (u + v)/2;
    printf("%f",av);
}
```

•**Assignment Operator:** +=, *=

1)

int i=2, x=3, y=4;

i=i+2; \leftrightarrow i += 2;

x =x*(y-1) \leftrightarrow x *= y-1;

2)

int d, e, f, g;

d -= 4; \leftrightarrow d = d -4;

e *= 5; \leftrightarrow e = e * 5;

f /= 3; \leftrightarrow f = f / 3;

g %= 9; \leftrightarrow g = g % 9;

Increment and Decrement Operators

•Increment operator (++)

–Can be used instead of **c+=1**

•Decrement operator (--)

–Can be used instead of **c-=1**

•Pre-increment

–Operator is used before the variable (**++c or --c**)

–Variable is changed before the expression it is in is evaluated

•Post-increment

–Operator is used after the variable (**c++or c--**)

–Expression executes before the variable is changed

Increment and Decrement Operators

- If c equals 5, then


```
printf("%d", ++c);
```

 – Prints 6

```
printf("%d", c++);
```

 – Prints 5
 – In either case, c now has the value of 6
- When variable not in an expression then
 - Preincrementing and postincrementing have the same effect


```
{++c; printf( "%d", c );}
```
 - Has the same effect as


```
{c++; printf( "%d", c );}
```
- Most binary operators (operators like + that have a left and right operand) have a corresponding assignment operator "op=", where "op" is one of

+ - * / % << >> & ^ |

Precedence and order of Evaluation of C operators

Operators	Associativity
() [] -> .	left to right
! ~ ++ -- + - * (type) sizeof	right to left
* / %	left to right
+ -	left to right
<< >>	left to right
< <= > >=	left to right
== !=	left to right
&	left to right
^	left to right
	left to right
&&	left to right
	left to right
? :	right to left
= += -= *= /= %= &= ^= = <<= >>=	right to left
,	left to right

Unary & +, -, and * have higher precedence than the binary forms

-4 * 3 < 1 && 2 > x + 5



((-4) * 3) < 1 && (2 > (x + 5))

-X^2*4+3>y^3-z/6*12-w



((-X)^(((2*4)+3)>y))^((3-((z/6)*12))-w)

Control Structures

Statement:

Simple statement:

A simple statement is just an expression followed by ; (semicolon).

e.g.

- `x = 100;`
- `f(x);`

Compound statement:

A sequence of statements enclosed in a pair of braces

e.g.

```
{  
  foo = x;  
  bar = y;  
}
```

Control flow statements:

- Normally statements are executed in the sequence in which they are given.

- What if we want to change ?

- Conditional statements
- Loop statements.

We will discuss it now!

Control Structure

- Sequential execution
 - Statements executed one after the other in the order written
- Transfer of control
 - When the next statement executed is not the next one in sequence
 - Overuse of `goto` statements led to many problems
- Bohm and Jacopini
 - All programs written in terms of 3 control structures
 - Sequence structures: Built into C. Programs executed sequentially by default
 - Selection structures: C has three types: `if`, `if/else`, and `switch`
 - Repetition structures: C has three types: `while`, `do/while` and `for`

Remark:

The structured program theorem, also called Böhm-Jacopini theorem, is a result in programming language theory.

It states that a class of control flow graphs (historically called charts in this context) can compute any computable function if it combines subprograms in only three specific ways (control structures).

These are

- Executing one subprogram, and then another subprogram (sequence)
- Executing one of two subprograms according to the value of a boolean expression (selection)
- Executing a subprogram until a boolean expression is true (iteration)

Control Structure

- Flowchart
 - Graphical representation of an algorithm
 - Drawn using certain special-purpose symbols connected by arrows called flowlines
 - Rectangle symbol (action symbol):
 - Indicates any type of action
 - Oval symbol:
 - Indicates the beginning or end of a program or a section of code
- Single-entry/single-exit control structures
 - Connect exit point of one control structure to entry point of the next (control-structure stacking)
 - Makes programs easy to build

The if Selection Structure

- Selection structure:
 - Used to choose among alternative courses of action
 - Pseudocode:
 - If student's marks is greater than or equal to 60*
 - Print "Passed"*
- If condition **true**
 - Print statement executed and program goes on to next statement
 - If **false**, print statement is ignored and the program goes onto the next statement
 - Indenting makes programs easier to read
 - C ignores whitespace characters

The if Selection Structure

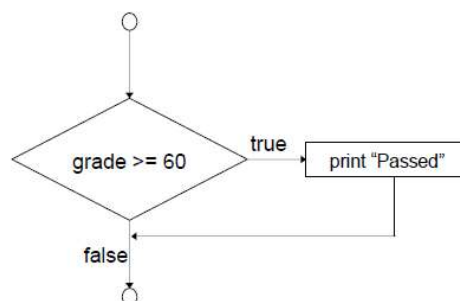
- Pseudocode statement in C:

```
if ( marks >= 60 )  
    printf( "Passed\n" );
```

 - C code corresponds closely to the pseudocode
- Diamond symbol (decision symbol)
 - Indicates decision is to be made
 - Contains an expression that can be **true** or **false**
 - Test the condition, follow appropriate path

The if Selection Structure

- **if** structure is a single-entry/single-exit structure



A decision can be made on any expression.

zero - **false**

nonzero - **true**

Example:

3 - 4 is **true**

The if/else Selection Structure

- **if**
 - Only performs an action if the condition is **true**
- **if/else**
 - Specifies an action to be performed both when the condition is **true** and when it is **false**
- Pseudocode:
 - If student's grade is greater than or equal to 60*
Print "Passed"
 - else*
Print "Failed"
 - Note spacing/indentation conventions

The if/else Selection Structure

- C code:

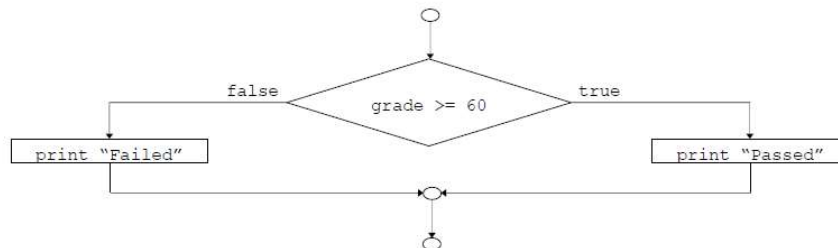

```
if ( grade >= 60 )
    printf( "Passed\n" );
else
    printf( "Failed\n" );
```
- Ternary conditional operator (?:)
 - Takes three arguments (condition, value if **true**, value if **false**)
 - Our pseudocode could be written:


```
printf( "%s\n", grade >= 60 ? "Passed" :
        "Failed" );
```
 - Or it could have been written:


```
grade >= 60 ? printf( "Passed\n" ) :
        printf( "Failed\n" );
```

The if/else Selection Structure

- Flow chart of the **if/else** selection structure



- Nested **if/else** structures
 - Test for multiple cases by placing **if/else** selection structures inside **if/else** selection structures
 - Once condition is met, rest of statements skipped
 - Deep indentation usually not used in practice

The if/else Selection Structure

- Pseudocode for a nested **if/else** structure

If student's grade is greater than or equal to 90

Print "A"

else

If student's grade is greater than or equal to 80

Print "B"

else

If student's grade is greater than or equal to 70

Print "C"

else

If student's grade is greater than or equal to 60

Print "D"

else

Print "F"

The if/else Selection Structure

- Compound statement:
 - Set of statements within a pair of braces
 - Example:


```
if ( grade >= 60 )
    printf( "Passed.\n" );
else {
    printf( "Failed.\n" );
    printf( "You must take this course
    again.\n" );
}
```
 - Without the braces, the statement


```
printf( "You must take this course
again.\n" );
```

 would be executed automatically

The if/else Selection Structure

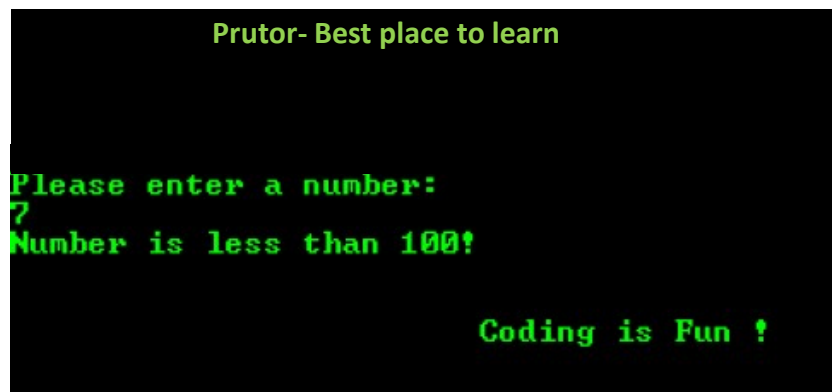
- Block:
 - Compound statements with declarations
- Syntax errors
 - Caught by compiler
- Logic errors:
 - Have their effect at execution time
 - Non-fatal: program runs, but has incorrect output
 - Fatal: program exits prematurely

A C-Program to discuss if-else control structure

```
#include<stdio.h>
int main()
{
    printf("\n\n\t\tPrutor- Best place to learn\n\n\n");
    int number;
    printf("Please enter a number:\n");
    scanf("%d",&number);
    /*
       For single statements we can skip the curly brackets
    */
    if(number < 100)
        printf("Number is less than 100!\n");
    else if(number == 100)
        printf("Number is 100!\n");
    else
        printf("Number is greater than 100!\n");
    printf("\n\n\t\tCoding is Fun !\n\n\n");

    return 0;
}
```

Output on Execution

A screenshot of a terminal window with a black background and green text. The output shows the program's execution: it first prints 'Prutor- Best place to learn' with two newlines and a tab. Then it prompts 'Please enter a number:' and the user has entered '7'. The program then prints 'Number is less than 100!' and finally 'Coding is Fun !' with two newlines and a tab.

```
Prutor- Best place to learn

Please enter a number:
7
Number is less than 100!

Coding is Fun !
```

```

/* Program to check whether the input integer number
 * is even or odd using the modulus operator (%)
 */

#include<stdio.h>
int main()
{
    int num;        // This variable is to store the input number
    printf("Enter an integer: ");
    scanf("%d",&num);

    // Modulus (%) returns remainder

    if ( num%2 == 0 ) printf("\n%d is an even number", num);
    else
        printf("\n%d is an odd number", num);
    return 0;
}

```

```

Enter a integer: 8
8 is an even number

```

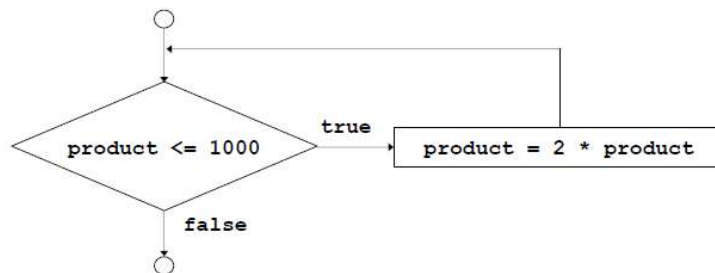
The while Repetition Structure

- Repetition structure
 - Programmer specifies an action to be repeated while some condition remains **true**
 - Psuedocode:
 - While there are more items on my shopping list*
 - Purchase next item and cross it off my list*
 - **while** loop repeated until condition becomes **false**

The while Repetition Structure

- Example:

```
int product = 2;
while ( product <= 1000 )
    product = 2 * product;
```



Formulating Algorithms

(Counter-Controlled Repetition)

- Counter-controlled repetition

- Loop repeated until counter reaches a certain value
- Definite repetition: number of repetitions is known
- Example: A class of ten students took a quiz. The grades (integers in the range 0 to 100) for this quiz are available to you. Determine the class average on the quiz
- Pseudocode:

Set total to zero

Set grade counter to one

While grade counter is less than or equal to ten

Input the next grade

Add the grade into the total

Add one to the grade counter

Set the class average to the total divided by ten

Print the class average

```
#include<stdio.h>
intmain()
{
    Int counter, grade, total, average;
    /* initialization phase */
    total = 0;
    counter = 1;
    /* processing phase */
    while( counter <= 10 )
    {printf( "Enter grade: " );
    scanf( "%d", &grade );
    total = total + grade;
    counter = counter + 1;
    }
    /* termination phase */
    average = total / 10;
    printf( "Class average is %d\n", average );
    return0;
    /* indicate program ended successfully */
}
```

Output of the Program

```
Enter grade: 98
Enter grade: 76
Enter grade: 71
Enter grade: 87
Enter grade: 83
Enter grade: 90
Enter grade: 57
Enter grade: 79
Enter grade: 82
Enter grade: 94
Class average is 81
```

Formulating Algorithms with Top-Down, Stepwise Refinement

- Problem becomes:
 - Develop a class-averaging program that will process an arbitrary number of grades each time the program is run.*
 - Unknown number of students
 - How will the program know to end?
- Use sentinel value
 - Also called signal value, dummy value, or flag value
 - Indicates “end of data entry.”
 - Loop ends when user inputs the sentinel value
 - Sentinel value chosen so it cannot be confused with a regular input (such as **-1** in this case)

Formulating Algorithms with Top-Down, Stepwise Refinement

- Top-down, stepwise refinement
 - Begin with a pseudocode representation of the *top*:
 - Determine the class average for the quiz*
 - Divide *top* into smaller tasks and list them in order:
 - Initialize variables*
 - Input, sum and count the quiz grades*
 - Calculate and print the class average*
- Many programs have three phases:
 - Initialization: initializes the program variables
 - Processing: inputs data values and adjusts program variables accordingly
 - Termination: calculates and prints the final results

Formulating Algorithms with Top-Down, Stepwise Refinement

- Refine the initialization phase from *Initialize variables* to:
 - Initialize total to zero*
 - Initialize counter to zero*
- Refine *Input, sum and count the quiz grades* to
 - Input the first grade (possibly the sentinel)*
 - While the user has not as yet entered the sentinel*
 - Add this grade into the running total*
 - Add one to the grade counter*
 - Input the next grade (possibly the sentinel)*

Formulating Algorithms with Top-Down, Stepwise Refinement

- Refine *Calculate and print the class average* to
 - If the counter is not equal to zero*
 - Set the average to the total divided by the counter*
 - Print the average*
 - else*
 - Print "No grades were entered"*

```

/* Average program with sentinel-controlled repetition */
#include<stdio.h>
Int main()
{
    Float average;
    /* new data type */
    Int counter, grade, total;
    /* initialization phase */
    total = 0;
    counter = 0;
    /* processing phase */
    printf( "Enter grade, -1 to end: " );
    scanf( "%d", &grade );
    while( grade != -1 )
    {
        total = total + grade;
        counter = counter + 1;
        printf( "Enter grade, -1 to end: " );
        scanf( "%d", &grade );
    }
    /* termination phase */
    if( counter != 0 )
    {
        average = ( float) total / counter;
        printf( "Class average is %.2f", average );
    }
    else
        printf( "No grades were entered\n" );
    return 0;
    /* indicate program ended successfully */
}

```

Output from the program:

```

Enter grade, -1 to end: 75
Enter grade, -1 to end: 94
Enter grade, -1 to end: 97
Enter grade, -1 to end: 88
Enter grade, -1 to end: 70
Enter grade, -1 to end: 64
Enter grade, -1 to end: 83
Enter grade, -1 to end: 89
Enter grade, -1 to end: -1
Class average is 82.50

```

Nested control structures

- Problem
 - A college has a list of test results (1 = pass, 2 = fail) for 10 students
 - Write a program that analyzes the results
 - If more than 8 students pass, print "Raise Tuition"
- Notice that
 - The program must process 10 test results
 - Counter-controlled loop will be used
 - Two counters can be used
 - One for number of passes, one for number of fails
 - Each test result is a number—either a 1 or a 2
 - If the number is not a 1, we assume that it is a 2

Nested control structures

- Top level outline
 - Analyze exam results and decide if tuition should be raised*
- First Refinement
 - Initialize variables*
 - Input the ten quiz grades and count passes and failures*
 - Print a summary of the exam results and decide if tuition should be raised*
- Refine *Initialize variables* to
 - Initialize passes to zero*
 - Initialize failures to zero*
 - Initialize student counter to one*

Nested control structures

- Refine *Input the ten quiz grades and count passes and failures* to

```

While student counter is less than or equal to ten
  Input the next exam result
  If the student passed
    Add one to passes
  else
    Add one to failures
  Add one to student counter
```

- Refine *Print a summary of the exam results and decide if tuition should be raised* to

```

Print the number of passes
Print the number of failures
If more than eight students passed
  Print "Raise tuition"
```

```

/* Analysis of examination results */

#include<stdio.h>
Int main()
{
  /* initializing variables in declarations */
  Int passes = 0, failures = 0, student = 1, result;

  /* process 10 students; counter-controlled loop */
  while( student <= 10 )
  {
    printf( "Enter result ( 1=pass,2=fail ): " );
    scanf( "%d", &result );
    if( result == 1 )          /* if/else nested in while */
      passes = passes + 1;
    else
      failures = failures + 1;
    student = student + 1;
  }
  printf( "Passed %d\n", passes );
  printf( "Failed %d\n", failures );
  if( passes > 8 )
    printf( "Raise tuition\n" );

  return 0;                  /* successful termination */
}
```

Program Output

```
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Passed 6
Failed 4
```