C Programming Essentials
Unit 2: Structured Programming

CHAPTER 4: STRUCTURED PROGRAMMING (DECISIONS)

DR. ERIC CHOU IEEE SENIOR MEMBER

LECTURE 1

Code Blocks (Basic project structure)

C Language/Program Components

Tokens: ID, keywords, variables. myVariable

Expression: lvalue, operator and/or operations : myVariable>=3, myVariable++

Statements: a complete program action: myVariable++;

if (myVariable>=3) System.out.println(myVariable);

Code Block: { statement1; statement2; statement3; }

Function: void function(){ statement1; statement2; statement3; }

Module (File): #include <stdio.h> void function(){ statement1; statement2; statement3; }

Project: overview.c, random.c, random.h, build.bat, random.exe

Putting the Main Program in a Separate File

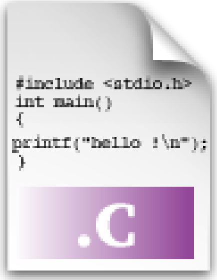
Demo Program: overview.c+random.c (overview project)

random.h

#include
#define
void funtion(void): function header
inline functions.

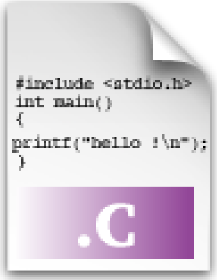
Static Binding
Header Needed

int main(void){
  return 0;
}



overview.c

extern void function(void); // importing



random.c

#include
#define
void funtion(void){
    // function implmentation
}

C++ implementation can also work similarly.

```java
public class Random
{
    public static double random(){
        return Math.random();
    }
    public static int randInt(int max){
        return (int) (Math.random()*max);
    }
    public static int randomInteger(int baseline, int steps, int count){
        return (int) (Math.random()*count) * steps + baseline;
    }
}
```

Dynamic Binding
No Header Needed

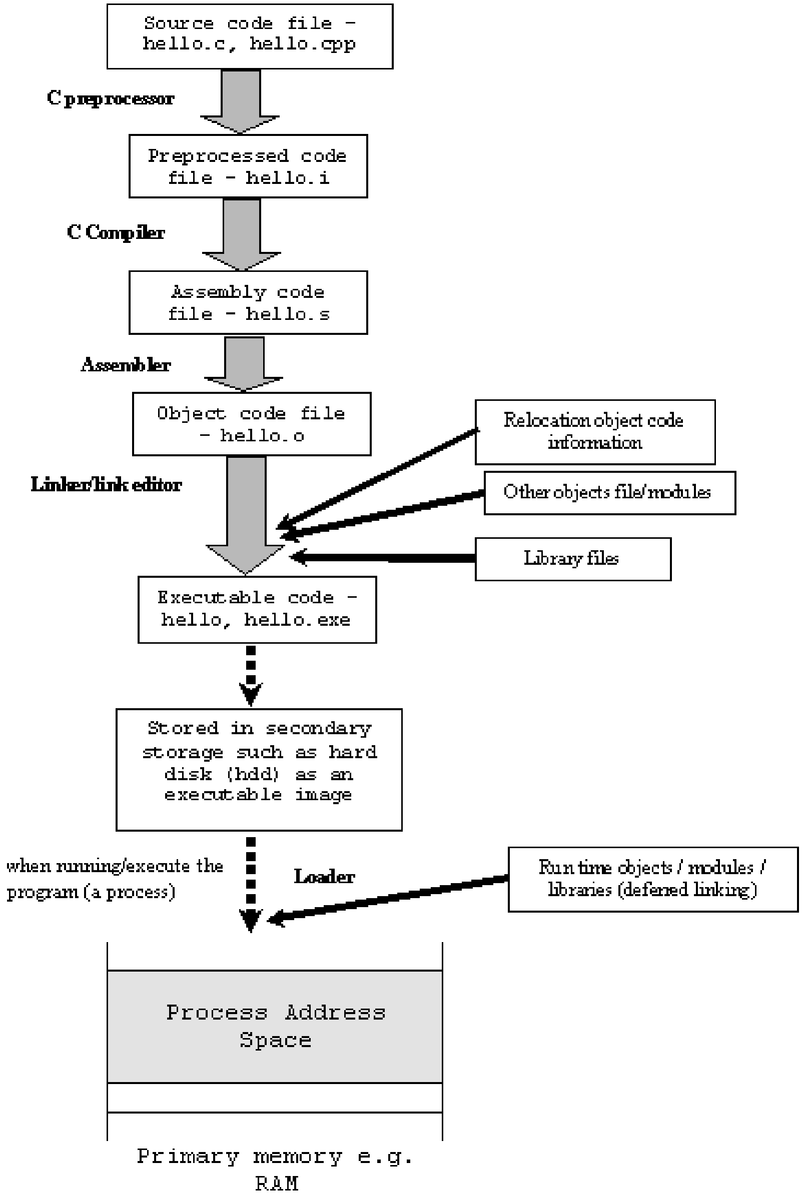
Random.java

Same package
implicit
importing

main()

```java
public class Overview
{
    final static int SAMPLES=10;

    public static void main(String[] args){
        int i=0;
        System.out.printf("\f");
        for (i=0; i<SAMPLES; i++){
            System.out.printf("Random Fraction=%6.4f\n", Random.random());
        }
        System.out.printf("\n");
        for (i=0; i<SAMPLES; i++){
            System.out.printf("Positive Integer (<10)=%d\n", Random.randInt(10));
        }
        System.out.printf("\n");
        for (i=0; i<SAMPLES; i++){
            System.out.printf("Random(+/- 5) =%d\n", Random.randomInteger(-5, 1, 11));
        }
    }
}
```

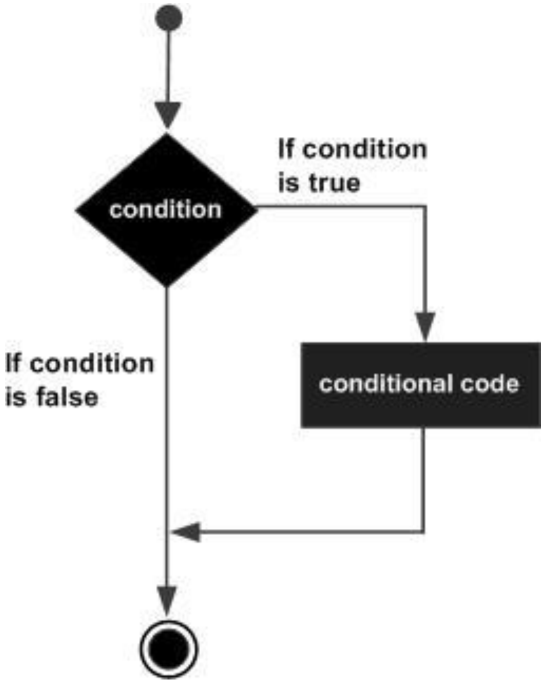
Overview.java


Source code file - hello.c, hello.cpp
C preprocessor
Preprocessed code file - hello.i
C Compiler
Assembly code file - hello.s
Assembler
Object code file - hello.o
Relocation object code information
Linker/link editor
Other objects file/modules
Library files
Executable code - hello, hello.exe
Stored in secondary storage such as hard disk (hdd) as an executable image
when running/execute the program (a process)
Loader
Run time objects / modules / libraries (deferred linking)
Process Address Space
Primary memory e.g. RAM

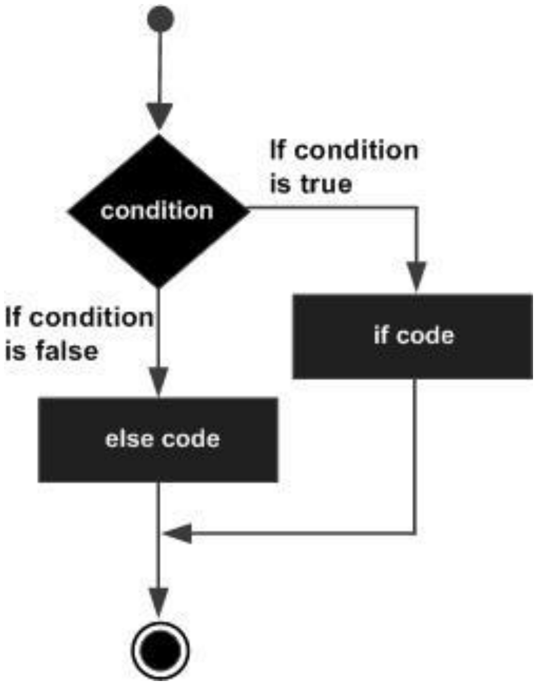
LECTURE 2

If-else statements

Decisions




```
if (condition){
    conditional code;
}
```




```
if (condition){
    if code;
}
else{
    else code;
}
```

condition is a logical expression

conditional code, if code, and else code are code blocks/statements

if-else statement

The syntax is

- if(expr) stmt
- if(expr) stmt1 else stmt2

Note that stmt, stmt1, stmt2 can either be simple or compound or control statements.

- Simple statement is of the form expr;
- Compound statement is of the form code blocks

```
{ stmt1;
  stmt2;
  ……….
  stmtn;
}
```

- Control Statement: will be discussed through this lecture. It involves if-else, for, switch, and etc.

e.g- if(expr) stmt1 else stmt2

Demo Program:
ifelse.c

Go gcc!!!

```
C:\Eric_Chou\C Course\C Programming Essentials\CDev\Ch4\ifelse>build

C:\Eric_Chou\C Course\C Programming Essentials\CDev\Ch4\ifelse>REM build if-else-statement example

C:\Eric_Chou\C Course\C Programming Essentials\CDev\Ch4\ifelse>gcc -std=c11 ifelse.c -o ifelse

C:\Eric_Chou\C Course\C Programming Essentials\CDev\Ch4\ifelse>ifelse
Enter an integer: 7
a is less than 20
value of a is : 7

C:\Eric_Chou\C Course\C Programming Essentials\CDev\Ch4\ifelse>ifelse
Enter an integer: 30
a is not less than 20
value of a is : 30
```

Expression Evaluation

Short-circuiting

- Consider `(a < b) && (b < c)`:
  - If **a >= b** there is no point evaluating whether **b < c** because **(a < b) && (b < c)** is automatically false
- Other similar situations

```
if (b != 0 && a/b == c) ...
if (*p && p->foo) ...
if (f || messy()) ...
```

- Can be avoided to allow for side effects in the condition functions

Demo Program:

short.c

Go gcc!!!

```
C:\Eric_Chou\C Course\C Programming Essentials\CDev\Ch4\short>short
Enter Two Numbers:
19
0
19.000000 0.000000
denominator is 0.0

C:\Eric_Chou\C Course\C Programming Essentials\CDev\Ch4\short>short
Enter Two Numbers:
19.0
1.0
19.000000 1.000000
a/b >= 3

C:\Eric_Chou\C Course\C Programming Essentials\CDev\Ch4\short>short
Enter Two Numbers:
19.0
-1.0
19.000000 -1.000000
a/b < 3.
```

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

int main(int argc, char *argv[]){
  double a;
  double b;

  printf("Enter Two Numbers: \n");
  scanf(" %lf", &a);
  scanf(" %lf", &b);
 printf("%f %f\n", a, b);
  if (b !=0.0 &&  a/b >=3) {
      printf("a/b >= 3 \n");
  }  else if (b ==0.0){
       printf("denominator is 0.0\n");
  } else{
      printf("a/b < 3.\n");
  }
  return EXIT_SUCCESS;
}
```

LECTURE 3

More on If-Else Statement

If – else

- The `if-else` statement expresses simplest decision making. The syntax is

```
if (expression)
    statement₁
else_opt
    Statement₂
```

- The `expression` is evaluated and if it is nonzero (true) then the $\texttt{statement}_1$ is executed. Otherwise, if else is present $\texttt{statement}_2$ is executed.
- `Expression` can just be numeric and need not be conditional expression only.
- $\texttt{Statement}_1$ and $\texttt{statement}_2$ may be compound statements or blocks.

Nested If-else-if-else statements

```
if (testExpression1) {
   // statements to be executed if testExpression1 is true
}
else if(testExpression2) {
   // statements to be executed if testExpression1 is false and
testExpression2 is true
}
else if (testExpression 3) {
   // statements to be executed if testExpression1 and
testExpression2 is false and testExpression3 is true
}
.
.
else {
   // statements to be executed if all test expressions are false
}
```

Demo Program: gpa.c

Go gcc!!!

```
if (score >= 90){
   grade = A;
} else if (score >= 80){
   grade = B;
} else  if (score >= 70){
   grade = C;
} else  if (score >= 60){
   grade = D;
} else {
   grade = F;
};
```

Right Version gpa.c

```
if (score < 60){
   grade = F;
} else if (score >= 60){
   grade = D;
} else  if (score >= 70){
   grade = C;
} else  if (score >= 80){
   grade = B;
} else {
   grade = A;
};
```

Wrong Version gpaBad.c

Dangling Else

The so-called "dangling else" problem is a perennial one that comes up when designing a programming language.

Given the code:

```
if (a)
if (b)
   s;
else
   t;
```

should it be parsed as:

```
if (a) {
  if (b)
    s;
  else
    t;
}
```

or as:

```
if (a) {
  if (b)
    s;
}
else
  t;
```

The usual (decades-old) solution for the compiler is to associate the else with the innermost if, and the ambiguity is resolved. End of story. (Associate with the closest if rule.)

But perhaps there's more to it.

It isn't ambiguous to the compiler. But it can be ambiguous to the programmer.

Consider:

```
if (a)
  if (b)
    s;
else
  t;
```

When the indentation looks like that, clearly the programmer is intending the else to be associated with the outer if, while the compiler will silently attach it to the inner if. (At least, in a language where whitespace indenting is irrelevant, like C, C++, Java, and D. This wouldn't be an issue in Python, which regards indenting as significant.)

if-else : Dangling Else

```
x = 1; y = 10;
if(y < 0)  if (y > 0) x = 3;
else x = 5;
printf("%d\n", x);
```

What is the output here?

```
if(z = y < 0) x = 10;

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

What is the output here?

if-else : Dangling Else

```
x = 1; y =10;
if(y < 0) if(y > 0) x = 3;
else x = 5;
printf("%d\n", x);
```

Output is : 1

Dangling else: else clause is always associated with the closest preceding unmatched if.

```
if(z = y < 0) x = 10;
printf("%d %d\n", x, z);
```

- The above code is equiv to the following one:

```
z = y <0;        // y<0 is false, then z = 0;
if (z) x = 10;   // if (0) x = 10;   // x remains 1
printf("%d %d\n", x ,z);
```

Output is: 1 0

Demo Program:

Dangling Else: dangling.c

Go gcc!!!

```c
int main(int argc, char *argv[]){
    int x = 1;
    int y =10;
    int z;
    if(y<0) if (y>0) x = 3;
    else x=5;
    printf("Output 1: %d\n", x);
    if(z = y < 0) x = 10;
    printf("Output 2: %d %d\n", x, z);
    return 0;
}
```

```
C:\Eric_Chou\C Course\C Programming Essentials\CDev\Ch4\dangling>dangling
Output 1: 1
Output 2: 1 0
```

LECTURE 4

Boolean Logic

Introduction

The most **obvious** way to **simplify** Boolean expressions is to manipulate them in the same way as normal **algebraic expressions** are manipulated. With regards to logic relations in digital forms, a set of rules for symbolic manipulation is needed in order to solve for the unknowns.

Introduction

A set of rules formulated by the English mathematician George Boole describe certain propositions whose outcome would be either true or false.

With regard to digital logic, these rules are used to describe circuits whose state can be either, 1 (**true**) or 0 (**false**). In order to fully understand this, the relation between the AND gate, OR gate and NOT gate operations should be appreciated. A number of rules can be derived from these relations as Table 1 demonstrates.

Introduction

A number of rules can be derived from these relations as Table 1 demonstrates.

- **Rule 1: X = 0 or X = 1**
- **Rule 2: 0 * 0 = 0**
- **Rule 3: 1 + 1 = 1**
- **Rule 4: 0 + 0 = 0**
- **Rule 5: 1 * 1 = 1**
- **Rule 6: 1 * 0 = 0 * 1 = 0**
- **Rule 7: 1 + 0 = 0 + 1 = 1**

Table 1: Boolean Postulates

Laws of Boolean Algebra

Table 2 shows the basic Boolean laws.

Note that every law has two expressions, (a) and (b). This is known as *duality*.

These are obtained by changing every AND(*) to OR(+), every OR(+) to AND(*) and all 1's to 0's and vice-versa.

It has become conventional to drop the ***** (AND symbol) i.e. A**.**B is written as AB.

Boolean Laws

T1 : Commutative Law

(a) $A + B = B + A$

(b) $A\,B = B\,A$

T2 : Associate Law

(a) $(A + B) + C = A + (B + C)$

(b) $(A\,B)\,C = A\,(B\,C)$

T3 : Distributive Law

(a) $A\,(B + C) = A\,B + A\,C$

(b) $A + (B\,C) = (A + B)\,(A + C)$

T4 : Identity Law

(a) $A + A = A$

(b) $A\,A = A$

T5 :

(a) $AB + A\overline{B} = A$

(b) $(A+B)(A+\overline{B}) = A$

T6 : Redundance Law

(a) $A + A\,B = A$

(b) $A\,(A + B) = A$

T7 :

(a) $0 + A = A$

(b) $0\,A = 0$

T8 :

(a) $1 + A = 1$

(b) $1\,A = A$

T9 :

(a) $\overline{A} + A = 1$

(b) $\overline{A}\,A = 0$

T10 :

(a) $A + \overline{A}\,B = A + B$

(b) $A\,(\overline{A} + B) = A\,B$

T11 : De Morgan's Theorem

(a) $(\overline{A + B}) = \overline{A}\;\overline{B}$

(b) $(\overline{A\,B}) = \overline{A} + \overline{B}$

<condition>: Boolean expressions

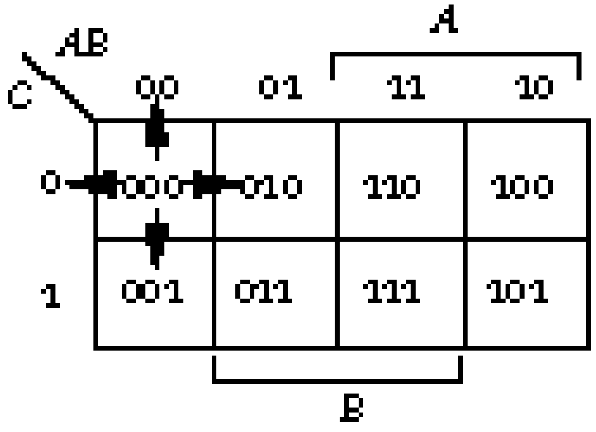
<expr A operator B> can have 16 possible outcomes which equals 2^4 (4 A/B combinations each has 2 possible outcome)

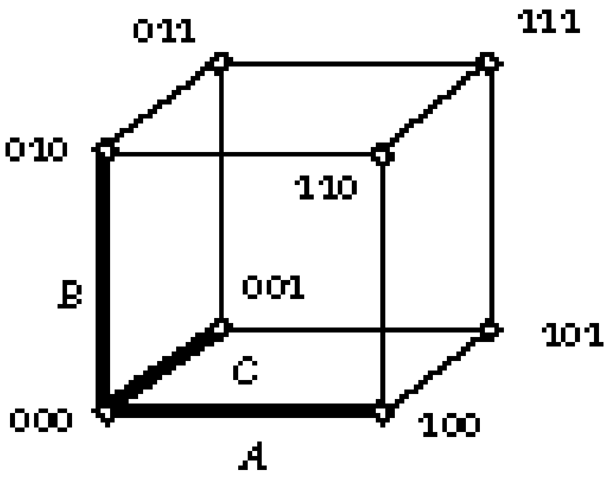
(Operators)

INPUT AB	ZERO	AND	A > B	A	B > A	B	XOR	OR	NOR	XNOR	NOT 'B'	A ≤ B	NOT 'A'	B ≤ A	NAND	ONE
00	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
01	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
10	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
11	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Boolean Cube and Theorem of Symmetry

The dual of any Boolean property





If Boolean Expression

T(A, B, +, *, =, 1, 0) is valid.

T(A, B , *, +, = 0, 1) is also valid.

A B + A = A

(A + B) * A = A

A + (-A) = 1

A * (-A) = 0

Example of De Morgan's Theorem

if ((x % 5 != 0) && (x % 2 != 0))   // x is not multiple of 5 and x is not multiple of 2.

equivalent to

if (!((x % 5 == 0) || (x % 2 == 0)))   // x is not (multiple of 5 or 2)

Download and work on BooleanQuiz.pdf

LECTURE 5

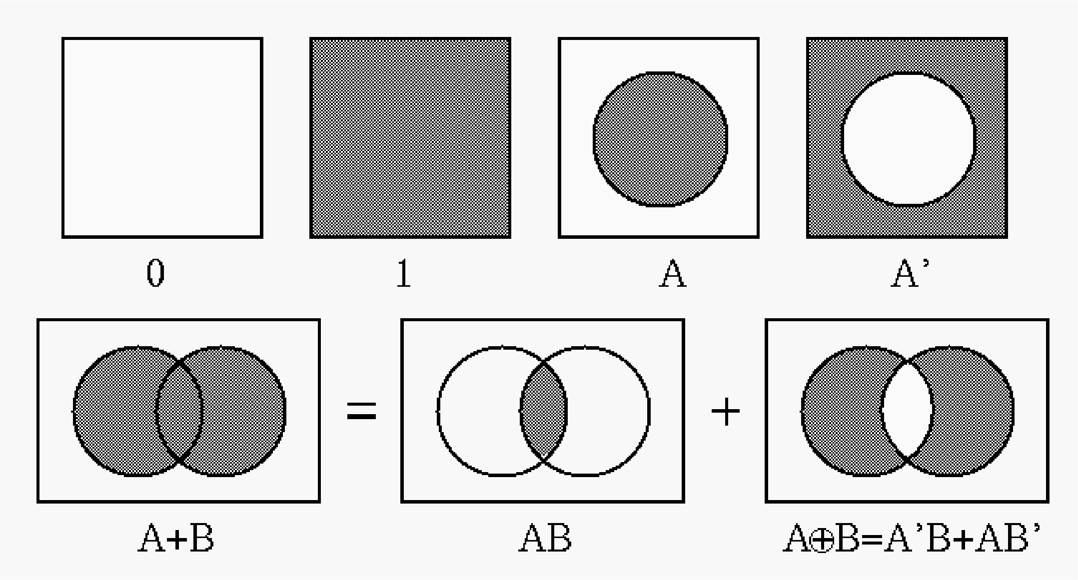
Logic Design

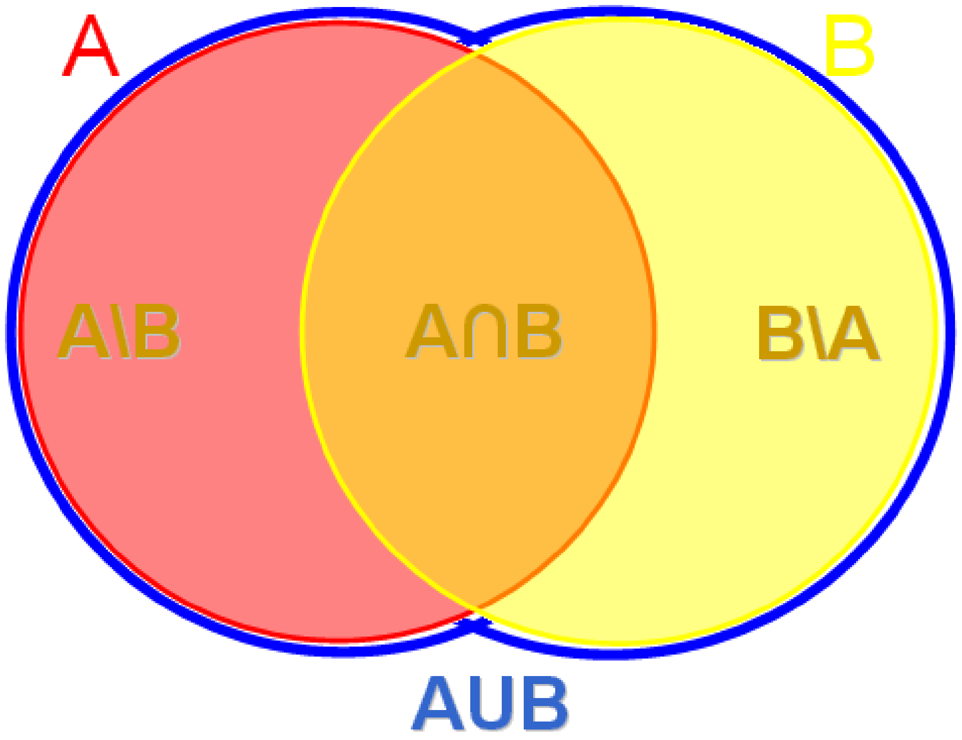
Set Theory and Logic Design (only positive numbers are shown)

bool m5 = (x % 5 == 0); // Set S5 = [0, 5, 10, 15, 20, …]

bool m2 = (x % 2 == 0); // Set S2 = [0, 2, 4, 6, 8, 10, …]

Boolean Expression	Set	Set Components
m5 && m2	$S5 \cap S2$	[0, 10, 20, 30, 40, …]
m5 || m2	$S5 \cup S2$	[0, 2, 4, 5, 6, 8, 10, 12, 14, 15, …]
m2 && !m5	$S2 - S5$	[2, 4, 6, 8, 12, 14, 16, 18, …]
m5 && !m2	$S5 - S2$	[5, 15, 25, 35, 45, …]
m2 ^ m5	$S5 \oplus S2$	[2,4, 5, 6, 8, 12, 14, 15, 16, …]



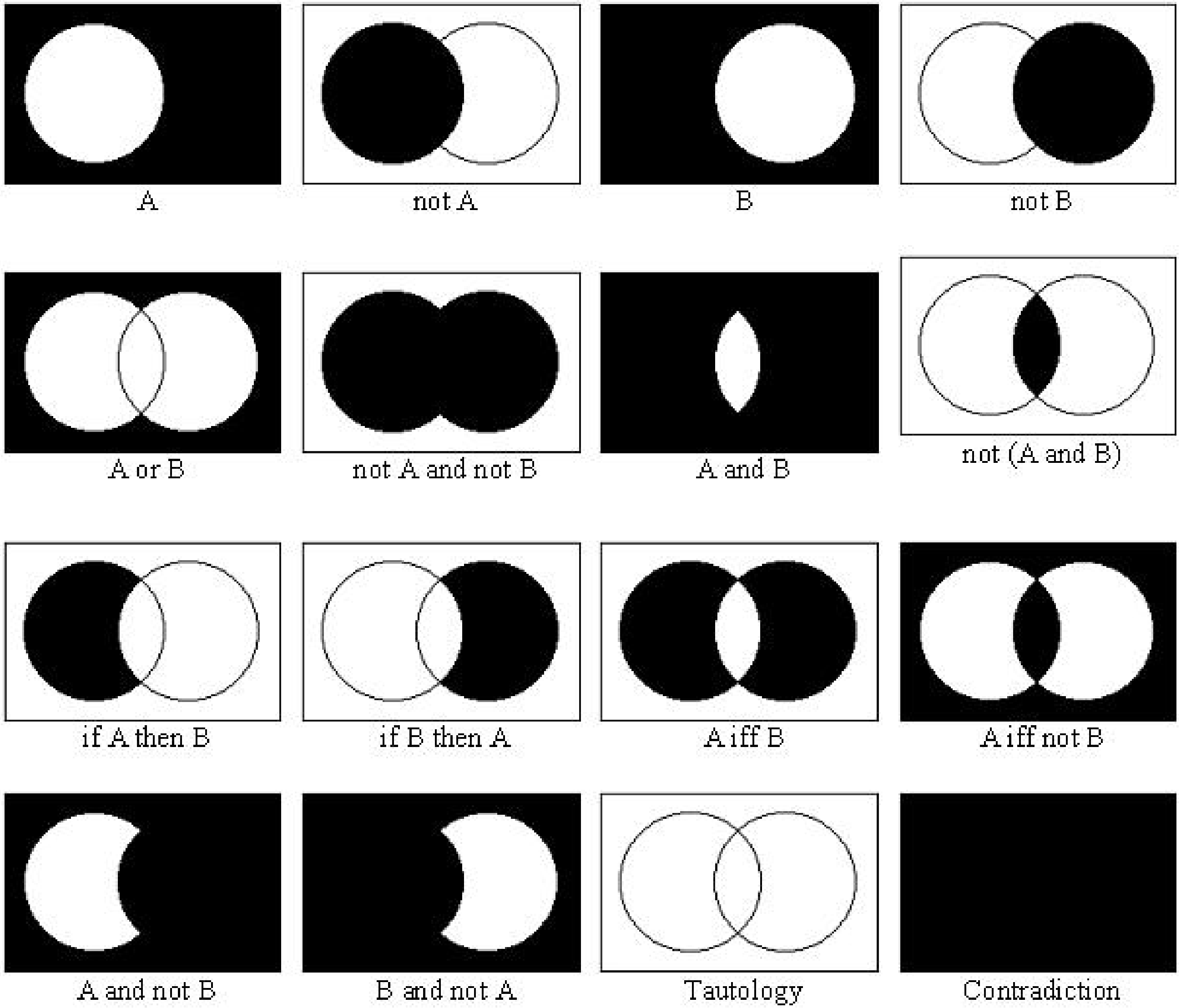


Venn Diagram Analysis

x	0	0	1	1
y	0	1	0	1
0	0	0	0	0
$x \cdot y$	0	0	0	1
$x \cdot y'$	0	0	1	0
x	0	0	1	1
$x' \cdot y$	0	1	0	0
y	0	1	0	1
$x \cdot y' + x' \cdot y$	0	1	1	0
$x + y$	0	1	1	1
$(x+y)'$	1	0	0	0
$x \cdot y + x' \cdot y'$	1	0	0	1
y'	1	0	1	0
$x + y'$	1	0	1	1
$x' \cdot y$	1	1	0	0
$x' + y$	1	1	0	1
$(x \cdot y)'$	1	1	1	0
1	1	1	1	1

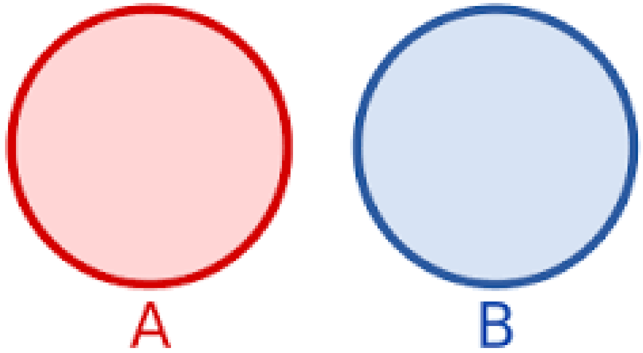
All 16 Boolean Functions can be expressed by (x, y, !, &&, ||)

Boolean Functions in Java	
bool f0 = false;	bool f8 = !(x+y);
bool f1 = x && y;	bool f9 = x && y + !x && !y;
bool f2 = x && !y;	bool f10 = !y;
bool f3 = x;	bool f11 = x + !y;
bool f4 = !x && y;	bool f12 = !x && y;
bool f5 = y;	bool f13 = !x + y;
bool f6 = x && !y + !x && y;	bool f14 = !(x && y)
bool f7 = x + y;	bool f15 = true;



Advanced Venn Diagram for all 16 Logic Functions for Two Inputs *f(A, B)*

Mutual Exclusive Sets



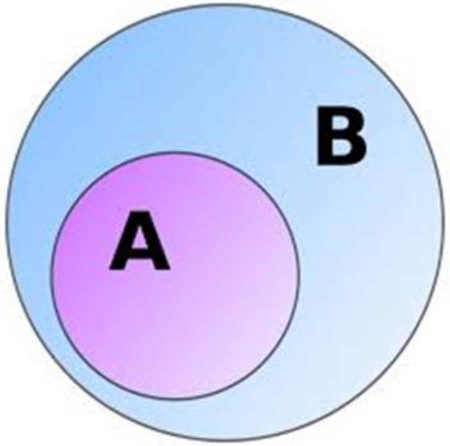

```
bool male   = (gender == 'M');
bool female = (gender == 'F');

if (male) { // all male get here , and all female will not get here}
if (female) { // all female get here, and all male will not get here}
```

Set Contained by Another Set

Contained by (A == (x>2), B == (x>1))

if a is true then b must be true   == !a || b

If x > 2 then x > 1 == !(x > 2) || (x>1)   (x>2) is contained by (x>1)



If you want to have A test first and B-A test:

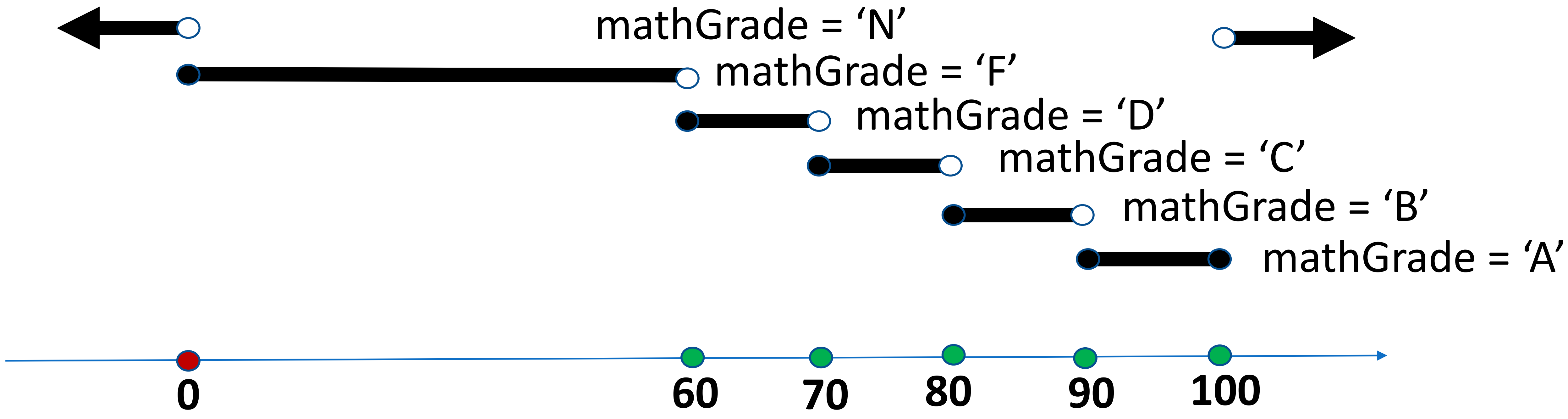
```
if (x > 2) { // all (x>2) get here }
else if (x > 1) { // only x>1 && !(x>2) get here }
```

If you want to have A test and then B test:

```
if (x>2) { // all x > 2 get here }
if (x>1) { // all x > 1 get here }
```

Don't try this: if (x>1) { // all x>1 get here } else if (x>2) { // no x can get here}

Number Line Analysis (Letter Grade)



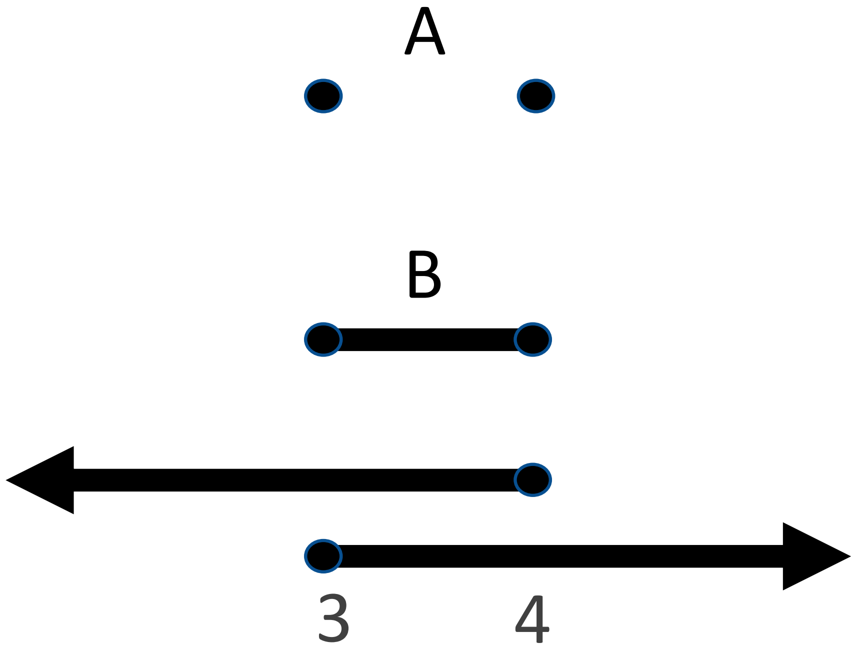
Sometimes the Logic Design Result Also Depends on the Data Type as Well

```
if (x == 3 ||  x == 4) { // set A }

if (x >= 3 && x <= 4) { // set B }
```

For int data type, A is equivalent to B

For double data type, A is not equivalent to B

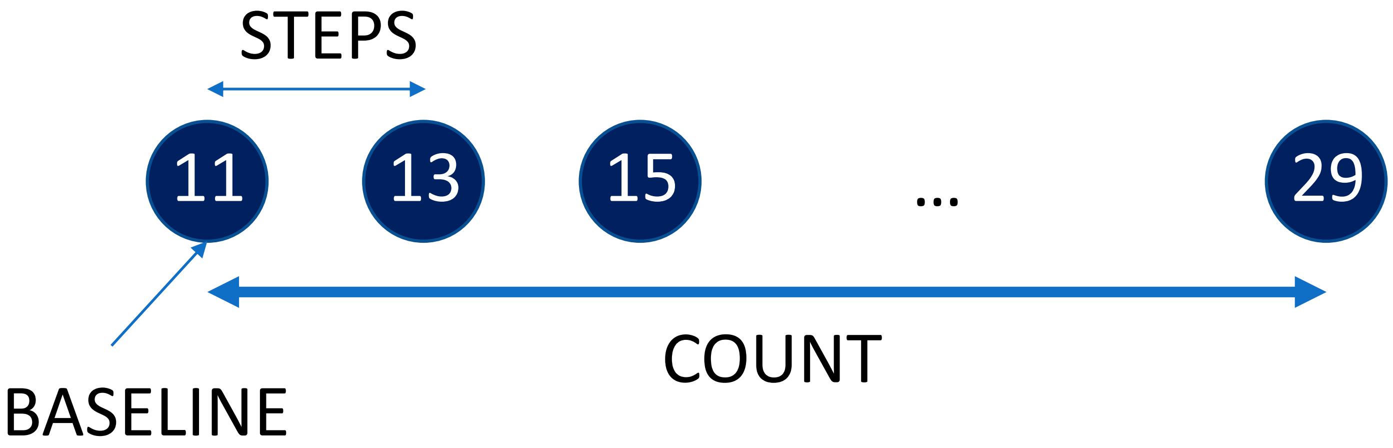
for example, x = 3.5 is in B but not in A



LECTURE 6

Random IF

Review of randomInteger(b, s, c)



Random Number Generation for Different Sample Regions **(weighted randomization)**

Generation Random Alphanumerical Symbols:

```
int region = (int) (random()* 3);
char dLetter = (char) (random()*10);
char aLetter = (char) (random()*26);
if (region == 0) aLetter = (char) (dLetter + ‘0’);
else if (region == 1) aLetter += ‘A’;
else aLetter += ‘a’;
// at the end, aLetter will be a random alphanumerical symbol.
```

Random Number Generation for Different Sample Regions (unweighted randomization)

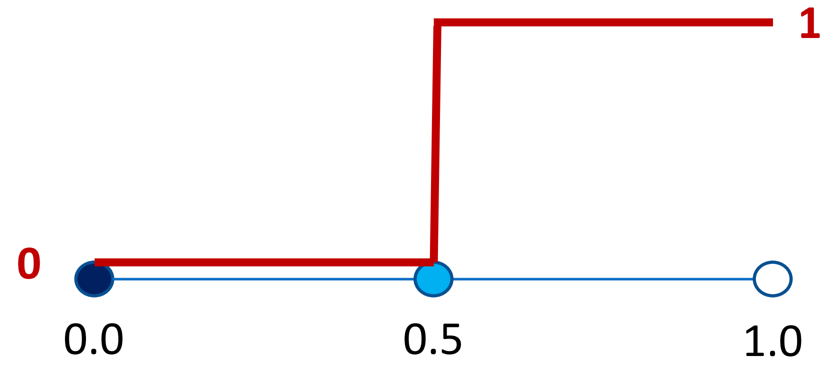
Generation Random Alphanumerical Symbols:

```
char aLetter = (char) (random()* 62);
if (aLetter <= 9) aLetter += '0';                                    // 0 to 9
else if (aLetter <= 35) aLetter = aLetter - (char) 10 + 'A'; // 10 - 35
else aLetter = aLetter – (char) 36 + 'a';                           // 36 – 61
// at the end, aLetter will be a random alphanumerical symbol.
```

Un-biased Randomized Coin (50-50)

Unbiased Random Number Generator:

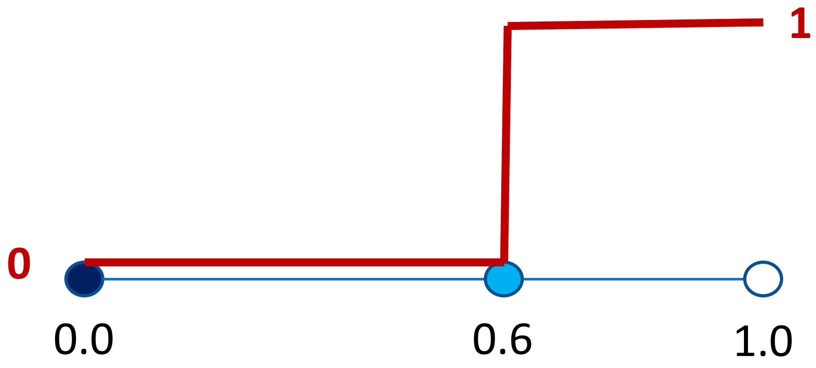
```
double randToss = random();

int die = 1;

if (randToss <= 0.5) die = 0;  // preset-else

// Think about it, you do not need the else-part.

// another way to write it.

int ide = (randToss<=0.5) ? 0 : 1;

// conditional expression.

//(coming lecture in this chapter)
```



Biased Randomized Coin (60% - 0 (Tail))

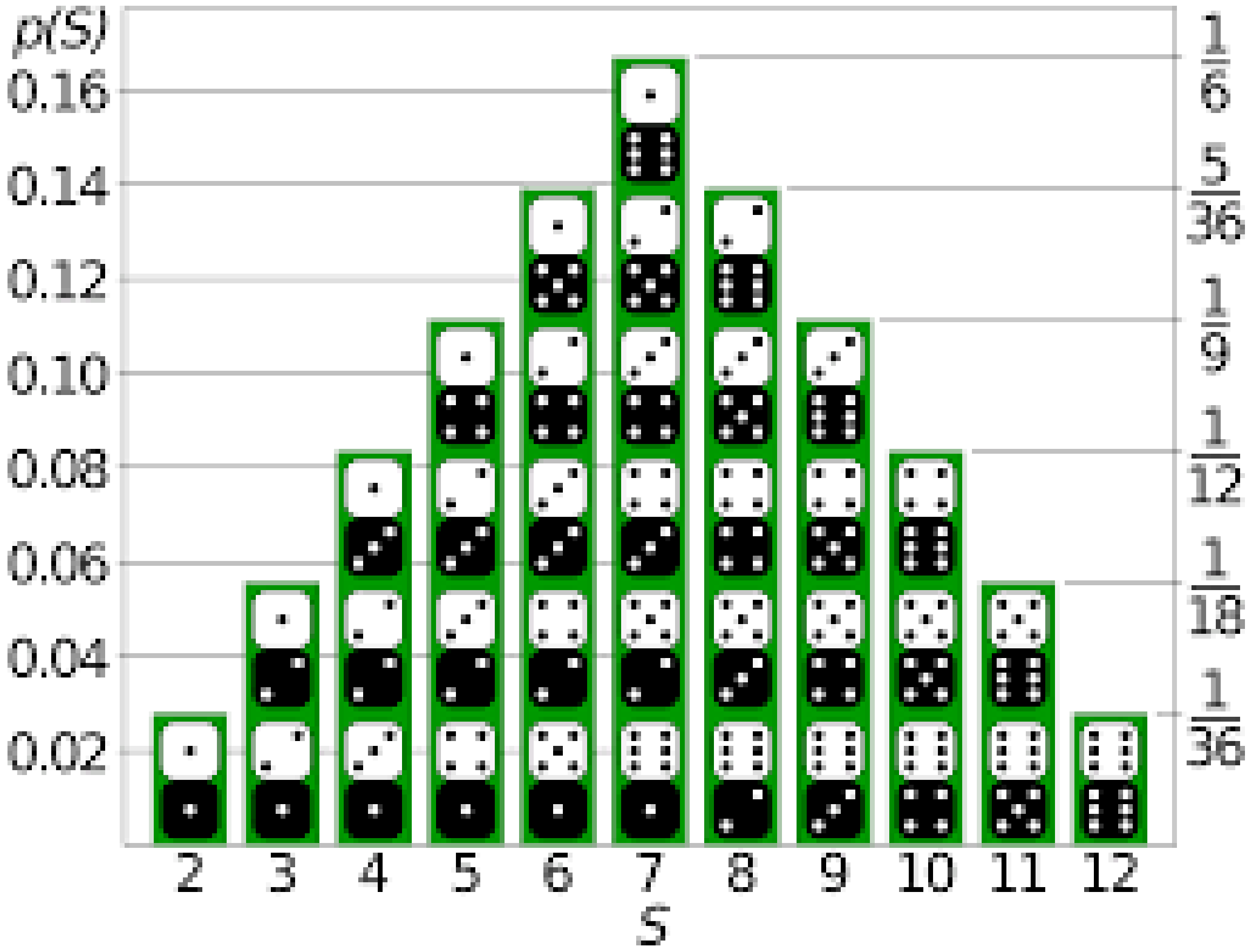
Unbiased Random Number Generator:

```
double randToss = random();
int die = 1;
if (randToss <= 0.6) die = 0;  // preset-else
// Think about it, you do not need the else-part.
// another way to write it.
int ide = (randToss<=0.6) ? 0 : 1;
// conditional expression.
//(coming lecture in this chapter)
```



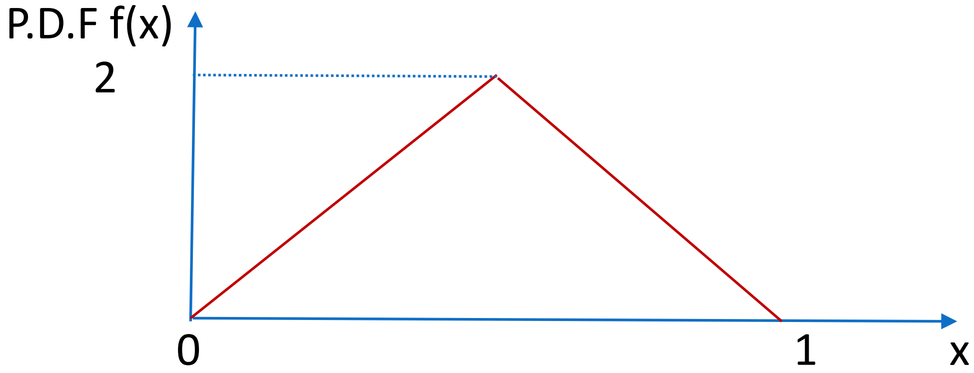
Sum of Two Dice Randomized Test

```
int die1 = (int) (random()*6) + 1;

int die2 = (int) (random()*6) + 1;

int sum = die1 + die2;
```



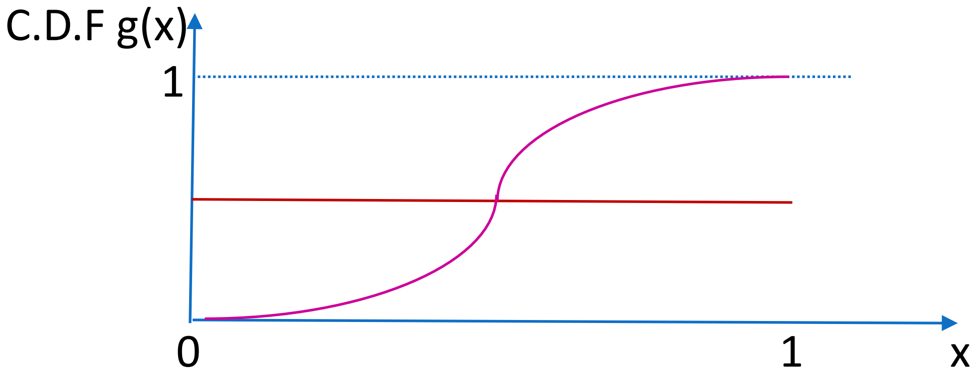
Biased Random Sample (Advanced Topic)

Using Cumulative Distribution Function for Ramping Distribution Samples.



$f(x) = 4 * x$ if $0 <= x < 1/2$;

$f(x) = - 4 * (x- 1)$ if $1/2<= x < 1$

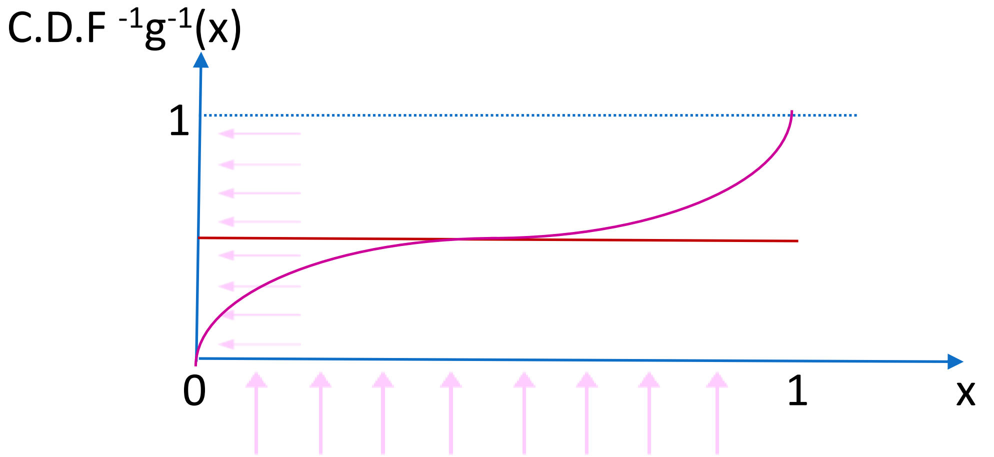


$g(x) = 2 * x^2$ if $0 <= x < 1/2$;

$g(x) = - 2(x -1)^2 + 1$ if $1/2<= x < 1$

Biased Random Sample (Advanced Topic)

Using Cumulative Distribution Function for Ramping Distribution Samples.



$g^{-1}(x) = (x/2)^{1/2}$ if $0 <= x < 1/2$;

$g^{-1}(x) = 1.0 - ((1-x)/2)^{1/2}$ if $1/2 <= x < 1$

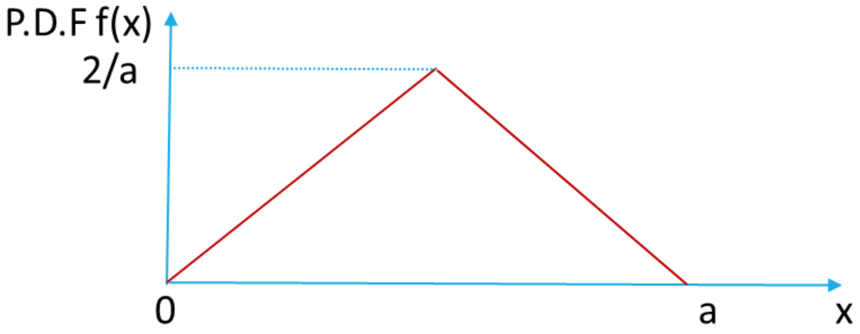
```
// Projection from random number
// generator to CDF-1 y = g-1(x),  y will have the
// probability distribution function of f(x) (PDF)
#include <math.h>

double x = random();
double y = 0.0;

if ( x>= 0 && x < 0.5) y = pow(x/2.0, 0.5);
else y = 1.0-pow((1-x)/2, 0.5);
```

Biased Random Sample (Advanced Topic)

Using Cumulative Distribution Function for Ramping Distribution Samples.

To generate random number of arbitrary range:

```
double z = y * SPAN + BASELINE;
```



$f(x) = 4/a^2 * x$ if $0 <= x < a/2$;
$f(f) = - 4/a^2 * (x- a)$ if $a/2<= x < a$;

This is using the same technique we used in generating a random sample over a certain range:

```
int r = (int) (random() * COUNT) * STEPS + BASELINE;
```

SPAN = 2.0, BASELINE = 3.0;

Run many times, the mean value is approaching BASELINE + SPAN/2.0;

BlueJ: Terminal Window - Chapter04

Options

```
X: 0.3606130321513341
Y: 0.4246251477193349 6
A Random Value: 3.84925029543867
Mean value after 10000 simulations: 3.9997526704840523
```

Why Random Number Generator is so Important in Computer Science?

Applications:

- Computer Game Design
- Monte Carlo Simulation (Computer Simulation)
- Use Simulation to Solve Hard Problems by Analytical Modelling
- Reality Check for a Project before Implementation

LECTURE 7

Switch

Switch

- The syntax of switch statement is

  switch (*expression*) {

  case *const-expression*$_1$: *statement*$_1$

  case *const-expression*$_2$: *statement*$_2$

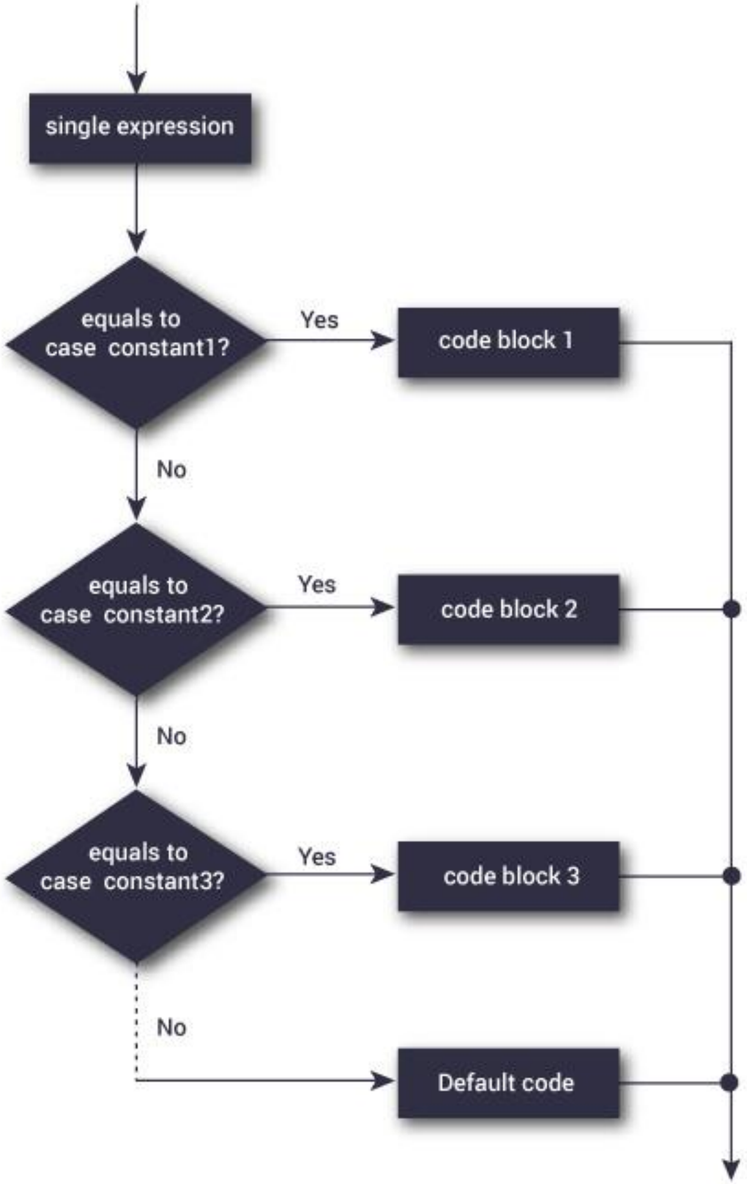
  :

  default : *statement*$_n$

  }
- All case expressions must be different. Expression must evaluate to an integer.
- First the expression is evaluated. Then the value of expression is compared with the case expressions. The execution begins at the case statement, whose case expression matches. All the statements below are executed.
- If default is present and if no other case matches then default statement is executed.

Switch Example

```
switch ( marks / 10 ) {
    case 3 : grade = “C”; break;
    case 4 : grade = “C+”; break;
    case 5 : grade = “B-”; break;
    case 6 : grade = “B”; break;
    case 7 : grade = “B+”; break;
    case 8 : grade = “A-”; break;
    case 9 :
    case 10 : grade = “A”; break;
    default : grade = “F”; break;
 }
```


single expression
equals to
case constant1?
Yes
code block 1
No
equals to
case constant2?
Yes
code block 2
No
equals to
case constant3?
Yes
code block 3
No
Default code

switch statement

Syntax is

- switch (expr) stmt
- expr must result in integer value; char can be used(ASCII integer value A-Z: 65-90, a-z: 97-122)
- stmt specifies alternate courses of action
  - case prefixes identify different groups of alternatives.
  - Each group of alternatives has the syntax

    case expr: stmt1; stmt; ……… stmtn; // thie case statement may be followed by a **break;** statement
  - Note that parentheses { } are not needed in case block
  - Multiple case labels

    case expr1: statements;

    case expr2: statements;

    … … …: statements;

    case exprn: statements;

switch statement as Token or Input Action Dispatching Unit

```
switch (letter = getchar()) {

 case‘a’: case ’A’: case ‘e’ : case ‘E’: case‘i’: case ‘I’: case ’o’ : case
 ‘O’: case‘u’: case ‘U’:

                printf(“Vowel”); break;   // category 1

   default:  printf(“Consonant”);     // category 2

 }
```

Note: the use of multiple cases for one group of alternative. Also note the use of default. Statement corresponding to default is always executed.

Demo Program:
switch.c

Go gcc!!!

LECTURE 8

Simulation Mode

Demo Program:
(SimulationMode.c)

Feature:

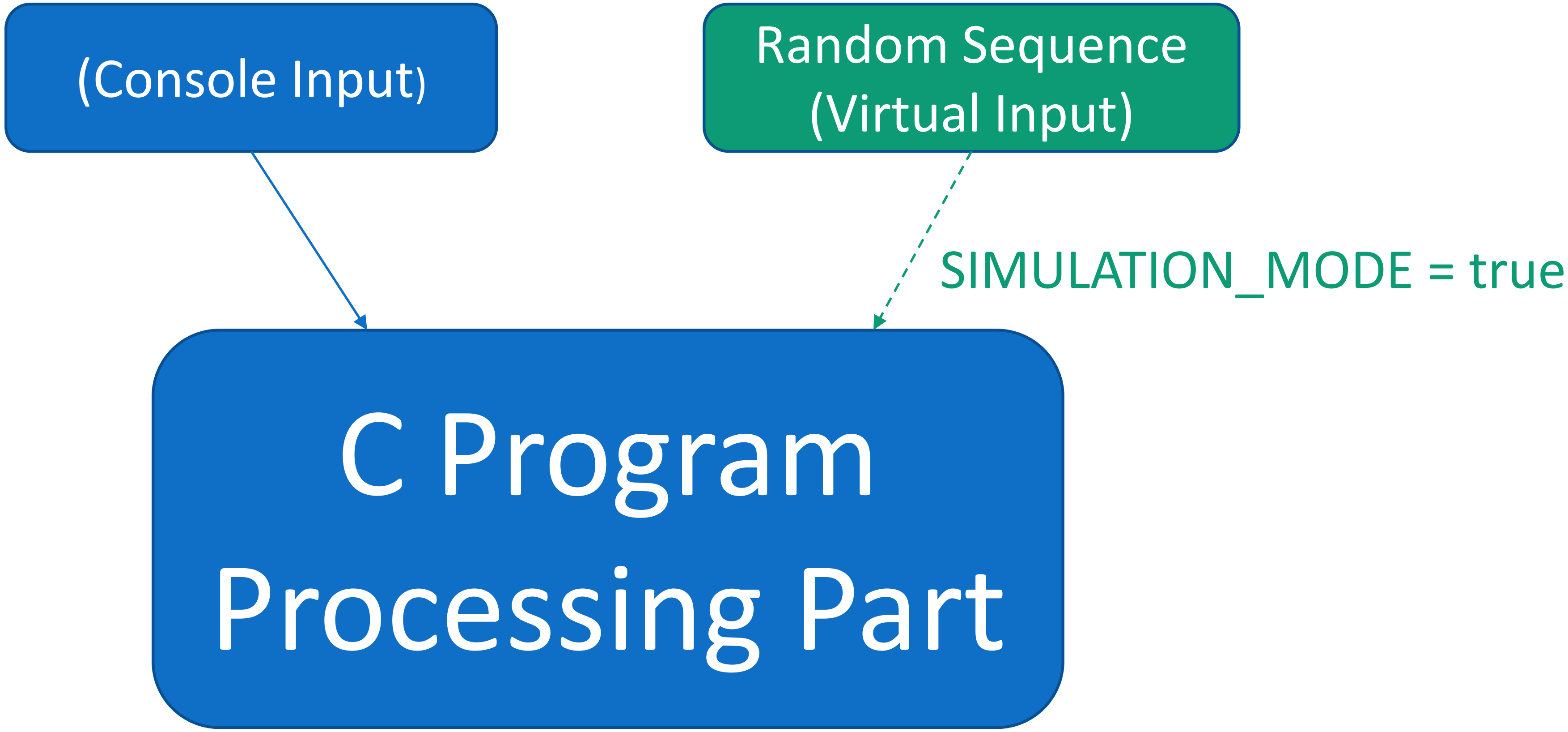
(1) Use Random class for random input for simulation mode.

(2) Random Seed for Random Number Generation.

(3) Update the Logic design for full coverage for score input domain to handle the abnormal inputs.

Random Source: Another Input Stream

(Not really I/O but for simulation mode)



Demo Program:
simulation.c + random.c

Go gcc!!!

Compiled in
Simulation Mode

```
C:\Eric_Chou\C Course\C Programming Essentials\CDev\Ch4\simulation>build

C:\Eric_Chou\C Course\C Programming Essentials\CDev\Ch4\simulation>REM build simulation example

C:\Eric_Chou\C Course\C Programming Essentials\CDev\Ch4\simulation>gcc simulation.c random.c -o simulation

C:\Eric_Chou\C Course\C Programming Essentials\CDev\Ch4\simulation>simulation
90.545654 - 84.024048 = 6.521606
C:\Eric_Chou\C Course\C Programming Essentials\CDev\Ch4\simulation>build

C:\Eric_Chou\C Course\C Programming Essentials\CDev\Ch4\simulation>REM build simulation example

C:\Eric_Chou\C Course\C Programming Essentials\CDev\Ch4\simulation>gcc simulation.c random.c -o simulation

C:\Eric_Chou\C Course\C Programming Essentials\CDev\Ch4\simulation>simulation
Enter an operator (+, -, *, /): +
Enter first operands: 87
Enter second operands: 28
87.000000 + 28.000000 = 115.000000
```