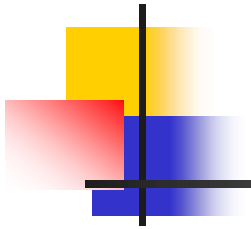


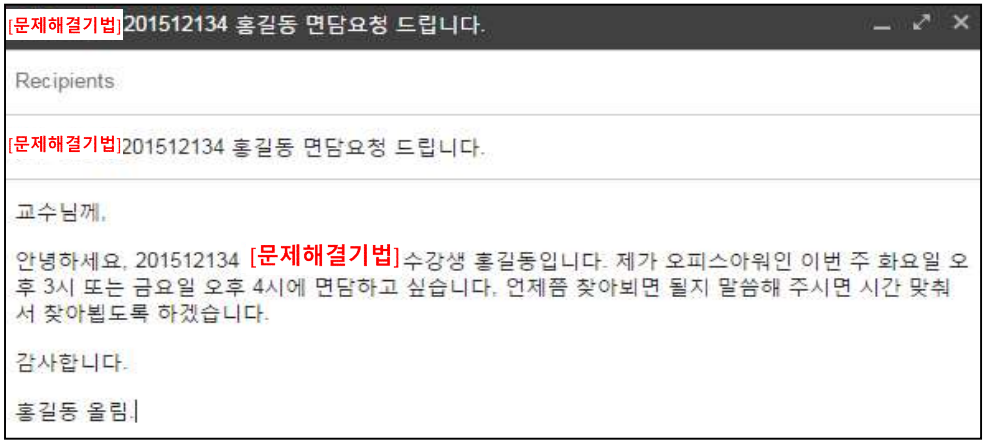
General Information

- Instructor
  - A/Prof. Jungchan Cho (조정찬)
  - Contact: AI관, #429,
  - thinkai@gachon.ac.kr
  - https://sites.google.com/view/visual-ai
  - Office hours: every Monday on appointment

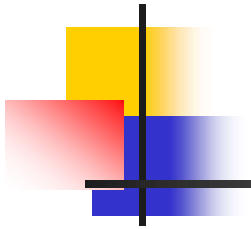
- Undergraduate TA
  - TBD (To-Be-Determined)
  - Office hours: TBA (Announced)

Contacting via E-mail

- **All contacts via E-mail: with [문제해결기법] header in mail title**



- **Contact TA first for lecture contents, assignment, etc..**
- **Use Lecturer for other Qs unable to solve with TA**

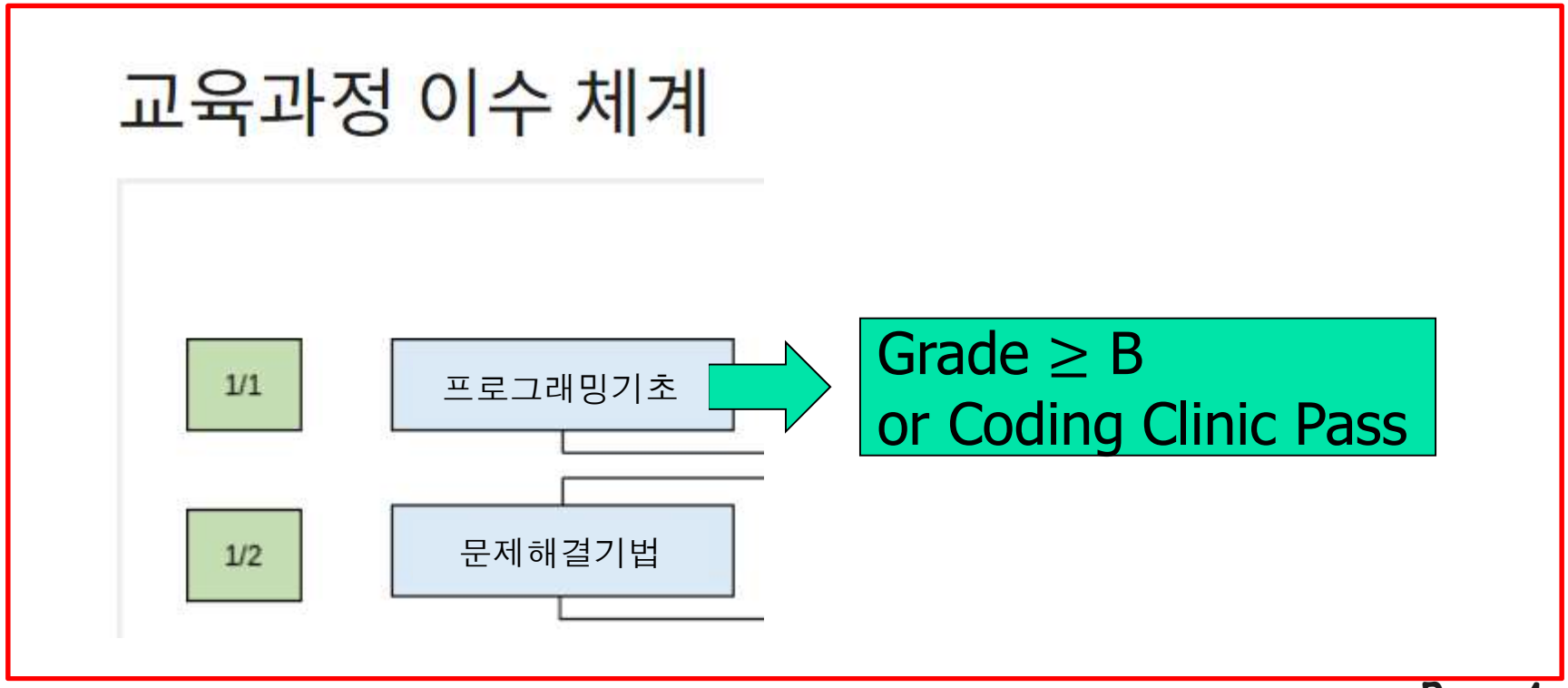


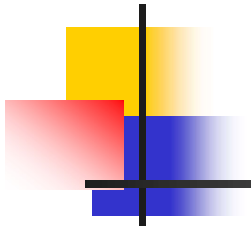
General Information

- This course is an ENGLISH-based lecture
  - Communication for lecture, Q/A, discussions can be with either English or Korean
  - The others including lecture slides, textbook, exams, quizzes, and assignments should be with English

Important Notice!!

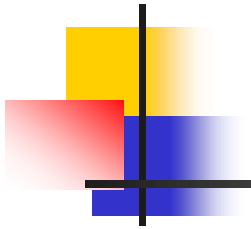
- Good programming skills in C (or C++)
- (almost) weekly homework and/or in-class exercise
- Prerequisite





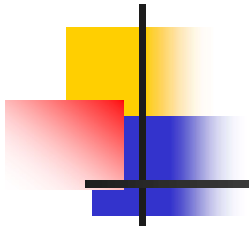
Textbook

- Not required
- Reference book
  - **"A First Book of" ANSI C**
    by Gary Bronson (Thomson)



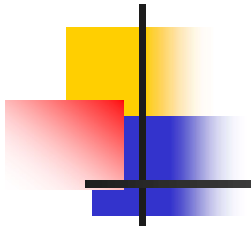
Course Rules (1/2)

- cheating in exams
  - will receive an "F" for the course
  - no scholarship for 4 years
- late homework
  - will receive a "0" point
- cheating on homework
  - will receive a "0" point
  - will receive a low semester grade



Course Rules (2/2)

- one "0" in programming homework:
  - course grade below "A"
- "under 50% in total homework points"
  - course grade below "B+"
- "not attending" 1/4 or more classes
  - course grade "F"
- "not attending" a class includes
  - being absent from or late to class
  - leaving the class in the middle
  - chatting in class
  - having the mobile phone on in class
  - coming to class in sandals or wearing a hat/cap or shorts



(tentative) Course Grading Policy

- attendance 10%
- homework: 20%
- mid-term exam: 20%
- final exam: 30%
- labs, term project, etc. : 20%

MOOC (massively online open courseware)

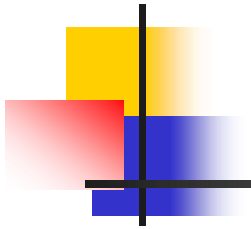
- At least 3 weeks' classes will be conducted using MOOCs
- Students can take the classes from anywhere any time (within one week of the posting of the MOOC)
- There would be exercise problems and assignments (due in one week)
- Students can ask questions online and receive answers within 24 hours
- This method has been used very successfully for 5 semesters for all SW ELITE courses and SW Basic courses for all non-Computer Science students in Gachon University

(tentative) Course Outline: Part 1

Week	Contents	Education
1st week (08/29~09/04)	- problem-solving	Lecture
2nd week (09/05~09/11)	- C review (arrays, strings, functions, pointers)	MOOC Lecture
3rd week (09/12~09/18)	- C review (struct, struct array)	MOOC Lecture
4th week (09/19~09/25)	- C review (struct, struct array) - lab 1 - patterns-1-struct-search	Lecture
5th week (09/26~10/02)	- patterns-1-struct-search	Lecture
6th week (10/03~10/09)	- patterns-1-struct-search	MOOC Lecture
7th week (10/10~10/16)	- patterns-2-recursion-update-copy-move - lab 2	MOOC Lecture
8th week (10/17~10/23)	- midterm exam	

(tentative) Course Outline: Part 2

Week	Contents	Education
9th week (10/24~10/30)	- patterns-3-fileIO-reorg-derivation - lab 3	Lecture
10th week (10/31~11/06)	- patterns-3-fileIO-reorg-derivation	Lecture
11th week (11/07~11/13)	- lab 4 - patterns-4-linkedList-search	Lecture
12 week (11/14~11/20)	- patterns-4-linkedList-search	Lecture
13th week (11/21~11/27)	- patterns-5-transform-bitwise - lab 5	MOOC Lecture
14th week (11/28~12/04)	- stack, queue	Lecture
15th week (12/05~12/11)	- final exam	Lecture

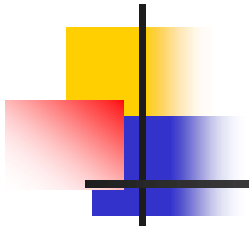


Special Requirement

- Bring a few sheets of A-4 paper to each class.
  - Those who do not bring their own sheets will receive penalty.

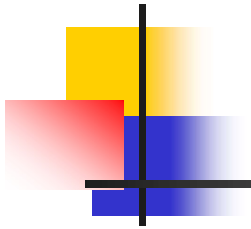
Program Patterns: Problem Solving Using C

Won Kim



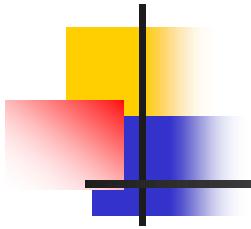
What This Course Is All About (1/3)

- The name of this course, when I first created it in 2010, was "Problem Solving Methods".
- The purpose of learning a programming language is to be able to solve many problems.
- The primary objective of this course is to teach problem solving skills using a programming language (whatever language).
- To solve problems using a programming language, you must first know how to use the language as a tool.
- It takes 2 to 3 semesters to properly learn a programming language like C, Java, Python.
- Once you know how to use C to solve problems, you can learn any other language quickly (in a few weeks).



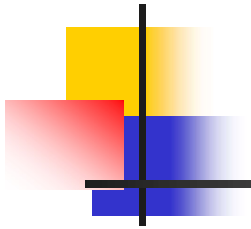
What This Course Is All About (2/3)

- Most of you knew nothing about programming when you joined our Department.
- We chose C as the first programming language, and designed a 3-course series to teach you how to solve problems using C.
- The C programming course you took last semester was the first exposure.
- This is the second of the three-course series.
- The third course is Data Structures (next semester).



What This Course Is All About (3/3)

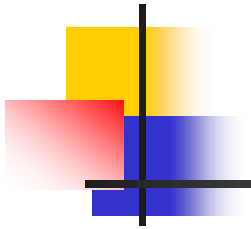
- To solve problems using a programming language takes more than just the language itself.
- You need several other skills.
  - You must understand most of the basic courses we require you for graduation. The first is the Data Structures course (next semester).
  - You need training in logical thinking.
  - You must understand and make use of software development methods.
  - You need to reuse code libraries (at a low level) and software design patterns (at a high level).



Summary of Course Objectives

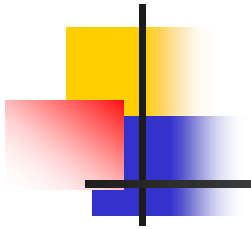
- Learn problem solving skills using programming.
- Review basic C
- Learn intermediate C.
- Learn basic data structures.
- Learn programming patterns.
- Learn software development methods

Course Outline



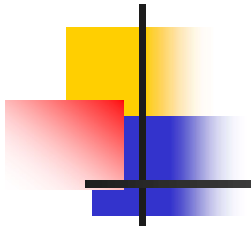
What You Need to Solve a Problem by Programming

- Problem solving skills
  - logical thinking
- Understanding a programming language
  - understanding all the key features of a language
  - correct syntax
  - debugging skills
- Programming skills
  - training on programming patterns
  - training on software engineering methods



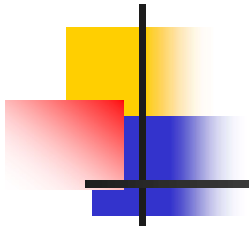
Logical Thinking

- Apply the step-by-step problem solving process
- Apply a divide and conquer strategy
  - Decompose a complex problem into small manageable pieces.
  - Solve the small problems and successively combine them into a total solution.



Intermediate C

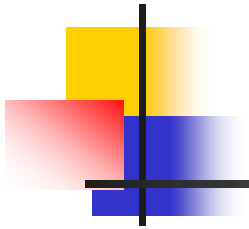
- file I/O
- dynamic memory allocation
- recursion
- bitwise operations
- basic data structures
  - struct, struct array
  - linked lists, stack and queue
- ** Note: The intermediate C features will be immediately put to solving problems. Just knowing the features and the syntax is not useful.



Software Engineering Methods

- Flow graph
- Documentation
- Coding guideline
- Software testing
- Debugging
- Software development process

Useful Programming Patterns



Note

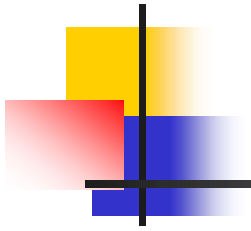
- The examples to follow are “low level” program patterns that often occur when you program.
- Knowing these will help you write more reliable programs and write programs faster. This is the case when you make use of built-in program libraries.
- The program patterns you will learn in this course are “high level” patterns.
- Knowing these high-level program patterns will help you significantly. It will help you design and code reliable programs and complete software development projects faster.

Testing Odd/Even Integer

```
int   num;

printf ("₩n Please type an integer ");
scanf ("%d ", &num);

if ((num%2)==1)
    printf ("\n an odd integer");
else
    printf ("\n an even integer");
```

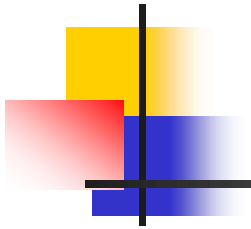
Testing Odd/Even Integer (another way)

```
int   num;

printf ("₩n Please type an integer ");
scanf ("%d ", &num);

if (num%2)
    printf ("\n an odd integer");
else
    printf ("\n an even integer");
```



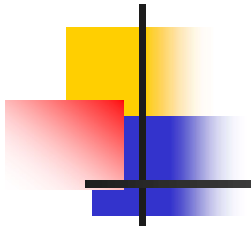
Exchanging Two Data

```
int   num1=100,   num2= 250;
```



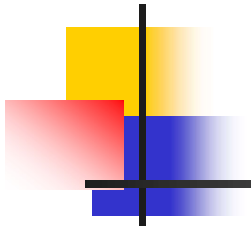
Use of a Temporary Variable

```
/*  exchanging two data  */

int  num1=100,  num2= 250;
int  temp;

temp = num1;
num1 = num2;
num2 = temp;
```



Finding Maximum/Minimum Data

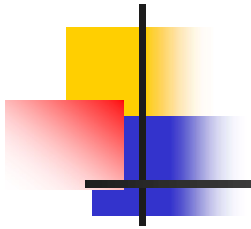
```
int   i, num;
int   max;

for (i=1; i<=100; i++)
{
    printf ("Please type an integer ");
    scanf ("%d ", &num);
    if (i==1)
        max = num;
    else if (num > max)
        max = num;
}
```



(High Level, Common) Program Patterns

- Data search
- Data update
- Data copying & moving
- Data transformation
- Data reorganization
- Data derivation

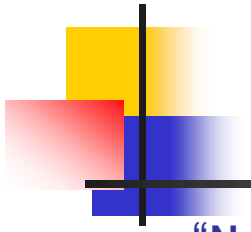
Problem Solving Method



Problem Solving Skills

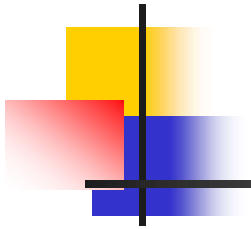
- Understand problem solving methods
- Use logical thinking
- Make use of programming patterns

- and Practice, Practice, Practice,…



Important to Take Into Account the Problem Size

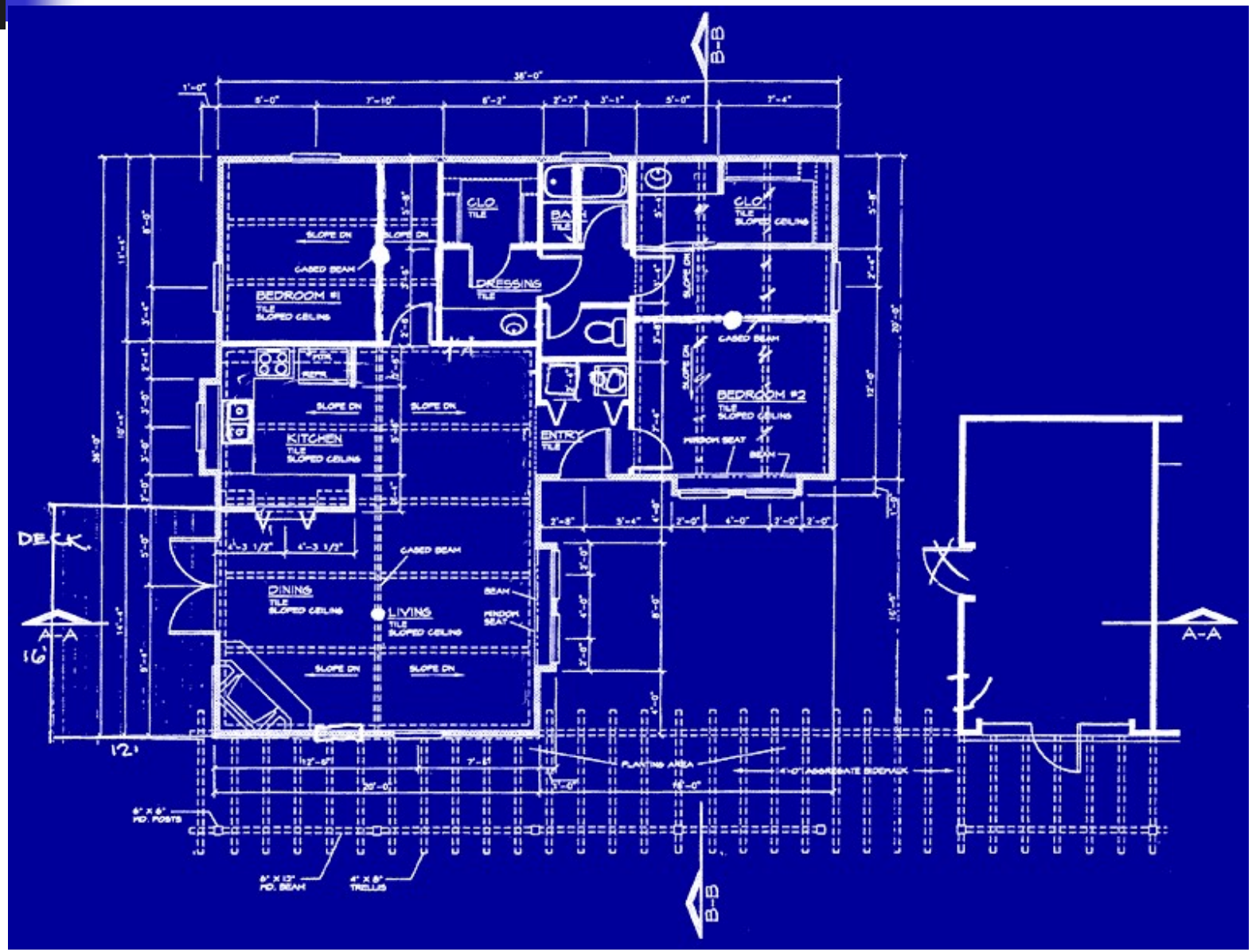
- "Nothing" (size)
  - examples in introductory C programming textbooks
  - exercises in "problem solving using C" course
- Tiny
  - exercises in "problem solving using C" course
  - team programming exercises in "problem solving using C" course
- Very Small
  - video rental management system, product defect detection system
- Small
  - VI editor
- Medium
  - Web browser, anti-virus software, C compiler, accounting software
- Large
  - word processor, email system, Web portal system, Internet search engine, enterprise application software, distributed online game
- Very Large
  - enterprise database management system, VoIP system
  - mobile phone software platform
- Huge
  - Windows 7 operating system



Problem Solving Methods

- "Nothing"
  - Just do it.
- Tiny
  - Needs logic formulation.
  - Mostly one solution.
- Very Small
  - Needs architecture formulation, and logic formulation.
  - There may be a few different solutions.

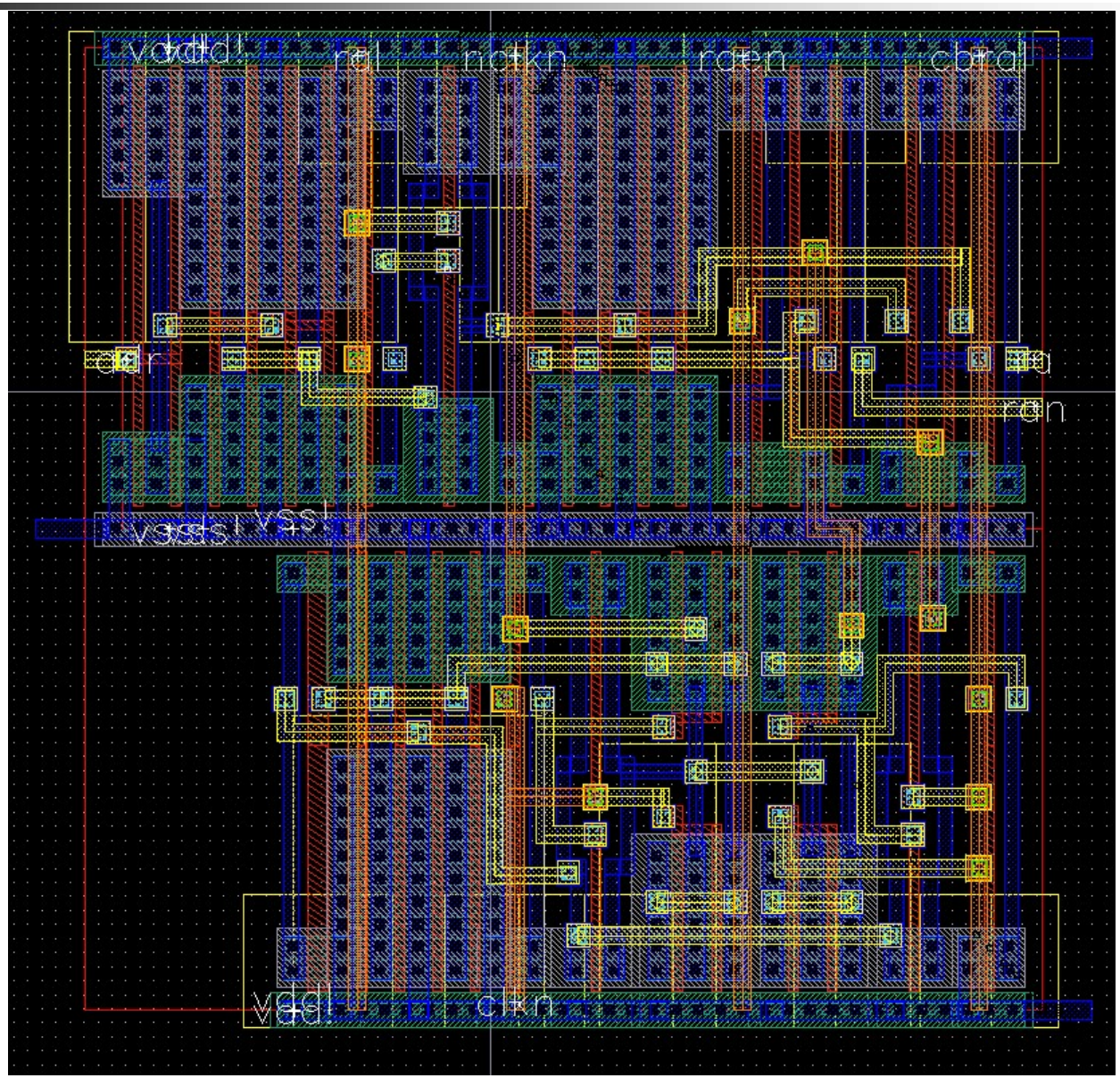
Building Blue Print -- what size?



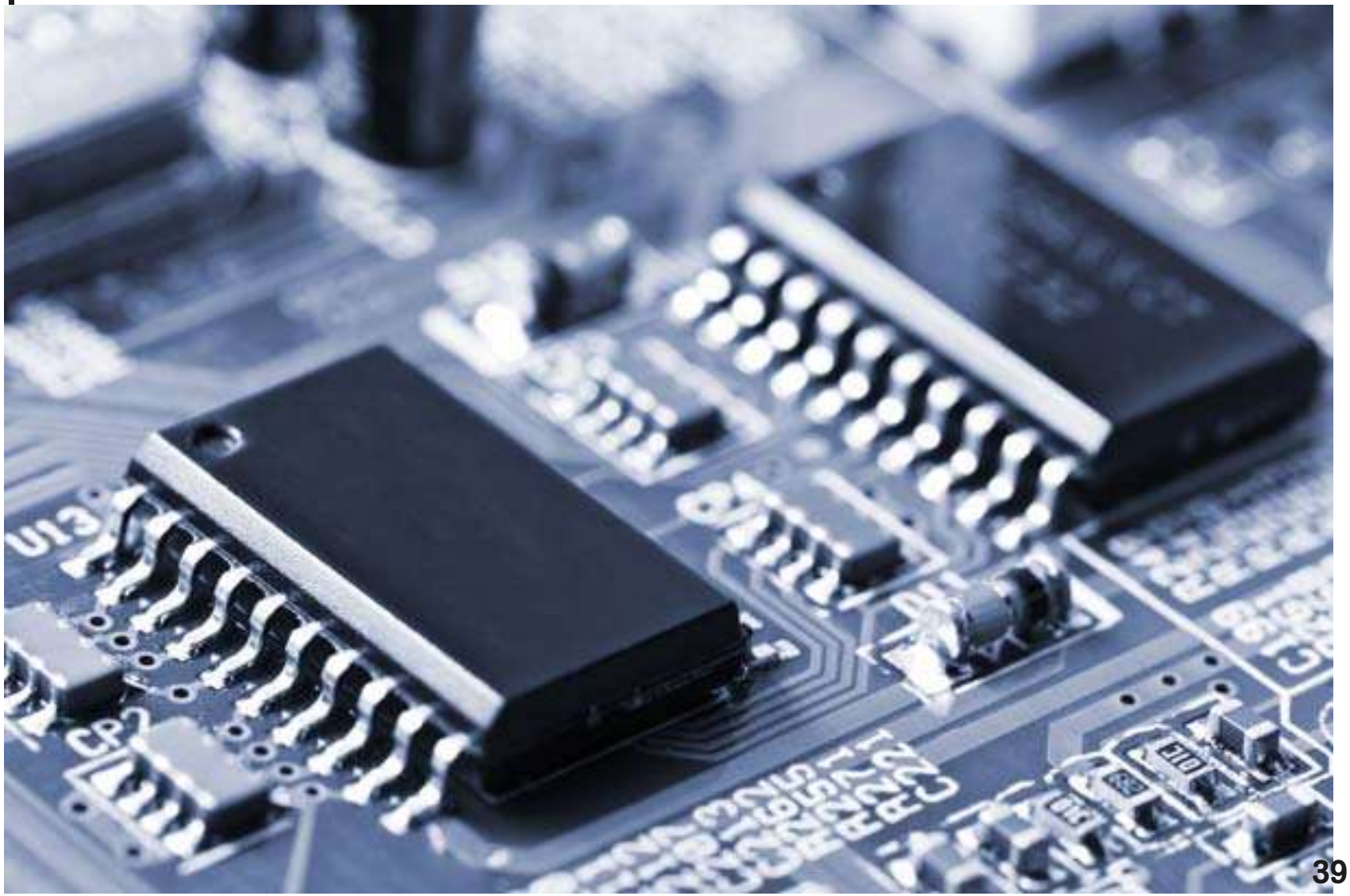
Sydney Opera House: a large scale building

Inside

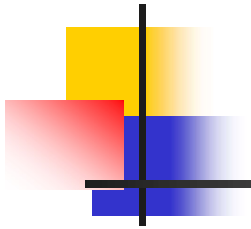
Semiconductor Design Layout - what size?



Semiconductor Chip

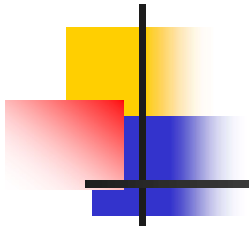


Computer Software Is the Most Complex Invention by Humans



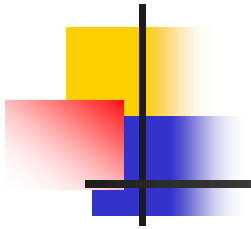
Require Software Development Process

- Requirements analysis
- Architecture (basic) design
- Test planning
- Detailed design – implementation – testing
- Documentation
- Maintenance



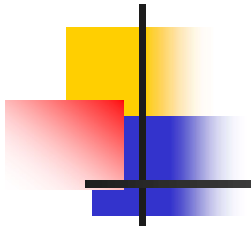
"Developing" a Program for Problem Solving (1/4)

- Step 1: understand the problem (requirements analysis)
  - Draw the problem.
  - Create simple examples.
- Step 2: outline a solution (basic design)
  - Draw the solution.
  - Validate it against the simple examples.
- Step 3: form a program structure (basic design)
  - Draw key elements of the structure.
  - Draw a flow graph.



"Developing" a Program (2/4)

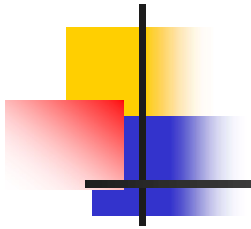
- Step 4: write a program outline (pseudo code) (detailed design)
  - Outline each key element of main.
  - Outline each function definition.
- Step 5: write the program (detailed design, implementation)
  - Complete each outline.
    - Follow coding guideline.
    - Follow coding style guideline.
  - Defensive coding.
  - Complete all declarations.
  - Complete all includes.



"Developing" a Program (3/4)

- Step 6: inspect the program (testing)
  - Draw memory diagrams for pointers.
  - Check loop, if else, switch, function.
  - Check variable initialization.
- Step 7: compile the program
  - Eliminate syntax errors.

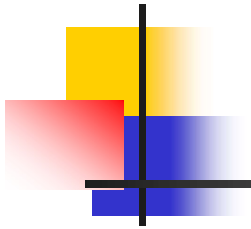
- Iterate all steps above, as necessary, for optimization and correctness.



“Developing” a Program (4/4)

- Step 8: test the program (testing)
  - Create test cases.
  - Run the program against the test cases.
  - Hand trace the program as needed.
- Step 9: document the program (documentation)
  - File comments
  - Key inline comments
- Step 10: maintain the program (maintenance)

(Logic) Flow Graph



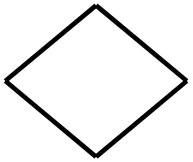
Flow Graph

- Describing the key elements of a program
  - pseudo code (using natural language phrases)
  - flow graph (flow chart) (using drawings)
- Flow graph
  - key components of a program
  - flow of control and data among them
- Different levels of detail
  - architecture diagrams of large software
  - basic design of a module
  - detailed design of a function
- Successive refinement

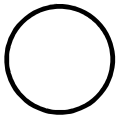
Key Symbols

action, module

flow line



decision



entry to/
exit from another part
of the flow graph

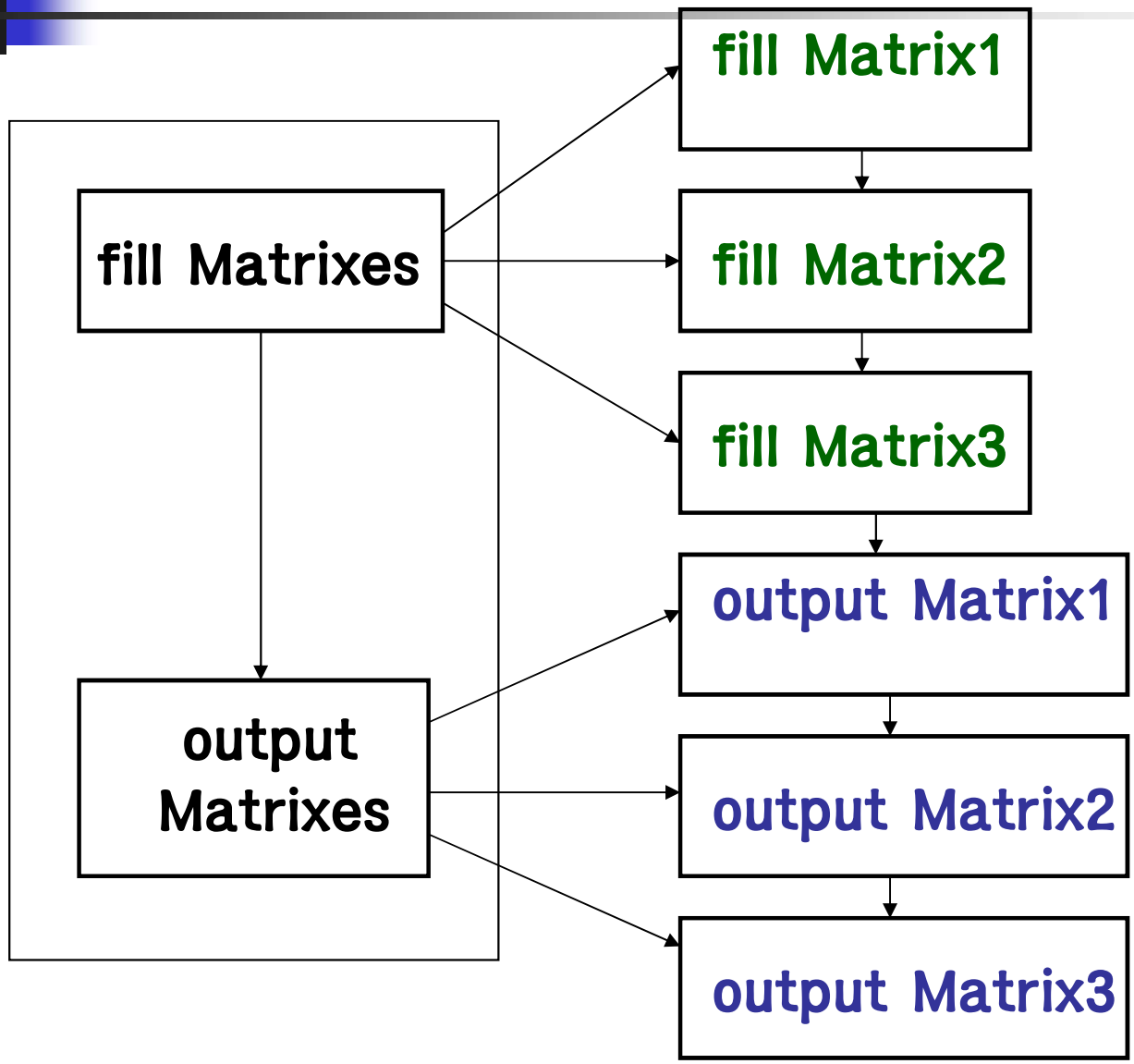
Example: Write a Problem for Matrix Computation

- Matrix Multiplication
  - Create 3 10x10 Matrixes
    - Matrix1: starting from 0, the value of each element increases by 1
    - Matrix2: starting from 0, the value of each element increases by 2
    - Matrix3: saves the result of multiplying Matrix1 and Matrix2
  - Output Matrix1, Matrix2, and Matrix3.
  - (Use 5 spaces, left flushed, for each element of Matrix1 and Matrix2; and 10 spaces for each element of Matrix3.)
- Expected Output:

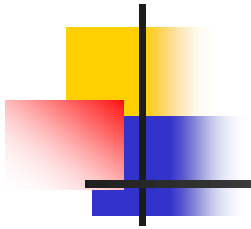
```
====================MATRIX1====================
0    1    2    3    4    5    6    7    8    9
10   11   12   13   14   15   16   17   18   19
20   21   22   23   24   25   26   27   28   29
30   31   32   33   34   35   36   37   38   39
40   41   42   43   44   45   46   47   48   49
50   51   52   53   54   55   56   57   58   59
60   61   62   63   64   65   66   67   68   69
70   71   72   73   74   75   76   77   78   79
80   81   82   83   84   85   86   87   88   89
90   91   92   93   94   95   96   97   98   99
```

```
====================MATRIX2====================
0    2    4    6    8    10   12   14   16   18
20   22   24   26   28   30   32   34   36   38
40   42   44   46   48   50   52   54   56   58
60   62   64   66   68   70   72   74   76   78
80   82   84   86   88   90   92   94   96   98
100  102  104  106  108  110  112  114  116  118
120  122  124  126  128  130  132  134  136  138
140  142  144  146  148  150  152  154  156  158
160  162  164  166  168  170  172  174  176  178
180  182  184  186  188  190  192  194  196  198
```

Flow Graph for the Example



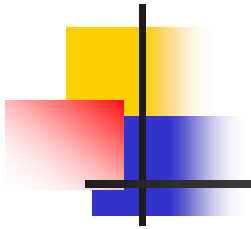
Problem Solving Approach: Tiny Example 1



A Tiny Problem

```
int  age[100], scores[100];
```

Given the ages and test scores of 100 students, write a C program that computes (& prints to the monitor) the ages of the students with the highest and lowest score, and the scores of the oldest and youngest students.



Understand the Problem

Draw the problem & create examples.

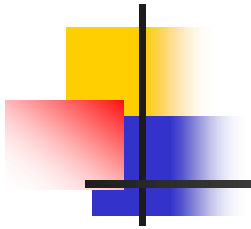
age	score
17	64

age	score
17	64
19	52

age	score
17	64
21	17
17	0
21	64

Identify cases to consider.

- multiple tie values
- multiple tie values spread out
- only one entry
- 0 score

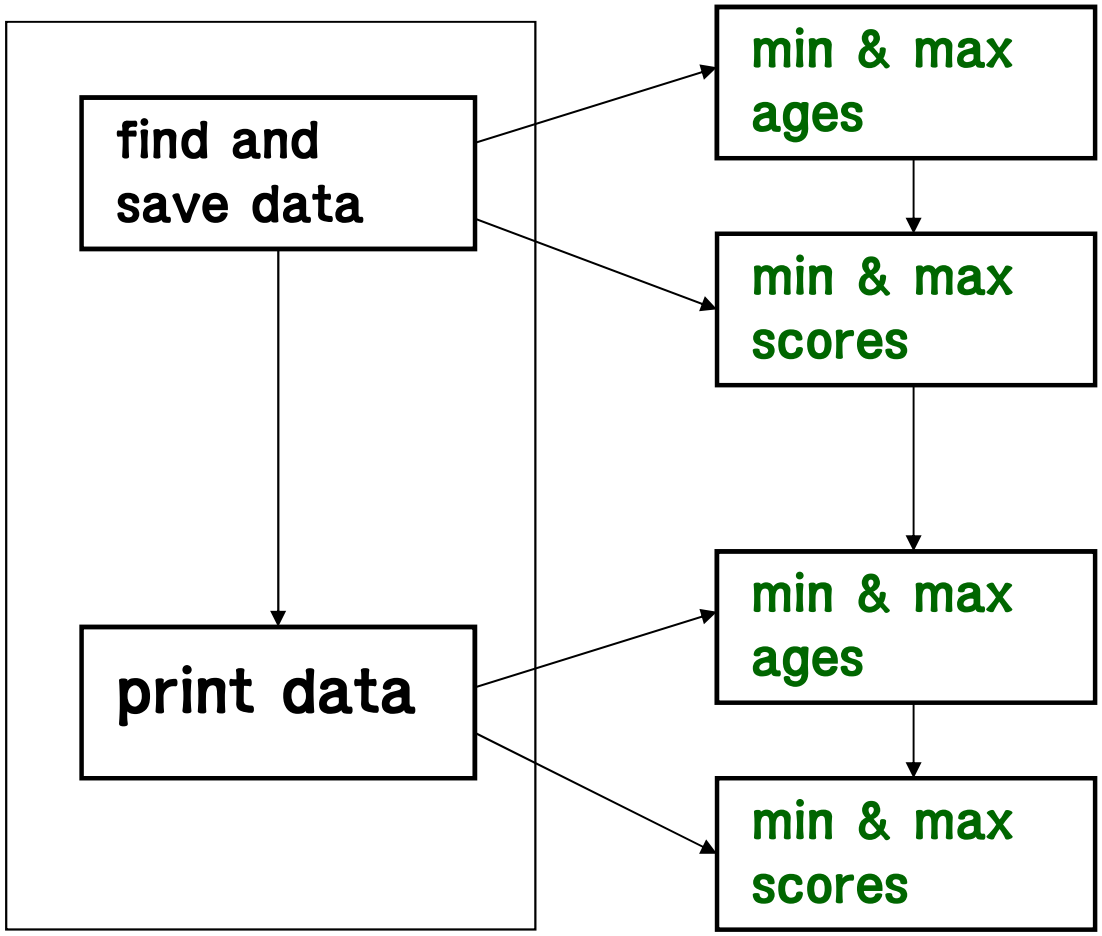


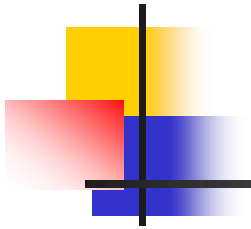
Outline a Solution

Loop through the age and score arrays
    find and save
        max age, min age, max score, min score
/* due to ties, save max/min data in arrays */

Loop through the saved arrays
    print scores (for max age, min age)
    print ages   (for max score, min score)

Flow Graph


find and
save data
print data
min & max
ages
min & max
scores
min & max
ages
min & max
scores



Form a Program Structure

```
declarations

main
    for loop (0 through 99)
        find & save max ages;
        find & save min ages;
        find & save max scores;
        find & save min scores;

    for loop (o through min/max counts)
        print scores for max age;
        print scores for min age;
        print ages for max score;
        print ages for min score;
```

Write a Program Outline: Pseudo Code

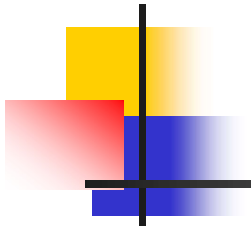
```
for (i=0; i<100; i=i+1) {
    if age[i] > maxage {
                        /* new max age found  */
       }
    else if age[i] = maxage {
                        /* tie max age found  */
       }

     …. /* for minage, maxscore, minscore */
    }


for (i=0; i <= maxagecnt; i=i+1)
    printf ("score for an oldest student = %d",  maxage array  );

…. /* print for min age, max score, min score arrays */
```

Write the Program – 1

```
for (i=0; i<100; i=i+1) {
    if (age[i] > maxage) {
        maxage = age[i];                    /* new max age found  */
        maxagecnt = 0;
        maxagegrp[maxagecnt] = i;
        }
    else if (age[i] == maxage) {
        maxagecnt = maxagecnt + 1; /* tie max age found  */
        maxagegrp[maxagecnt] = i;
        }

     ....
    }

for (i=0; i <= maxagecnt; i=i+1)
    printf ("score for an oldest student = %d", score[maxagegrp[i]]);

....
```

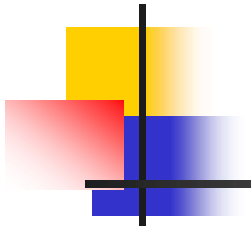
Write the Program – 2 (1/2)

```
int maxage=-1, minage=1000;
int maxscore=-1, minscore=1000;
int maxagecnt, minagecnt, maxscorecnt, minscorecnt;
int age[100], maxagegrp[100], minagegrp[100];
int score[100]; maxscoregrp[100], minscoregrp[100];
int i;

for (i=0; i<100; i=i+1) {
   if (age[i] > maxage) {
      maxage = age[i];              /* new max age found  */
      maxagecnt = 0;
      maxagegrp[maxagecnt] = i;  /* save the age index  */
      }
   else if (age[i] == maxage) {
      maxagecnt = maxagecnt + 1; /* tie max age found  */
      maxagegrp[maxagecnt] = i;  /* save the age index  */
      }

    …. /* for minage, maxscore, minscore */
   }
```



Write the Program – 2 (2/2)

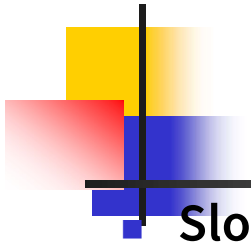
```
for (i=0; i <= maxagecnt; i=i+1)
    printf ("score for an oldest student = %d", score[maxagegrp[i]]);

…. /* print for maxscoregrp, minscoregrp */
```



Excercise

- Complete the example program and run it, using two different sets of data.
- (* Make each array a 5-element integer array.)
- (* Enter some “-1” in the array. Be careful with your ‘find max” or “find min” code.)

Problem Solving Approach: Tiny Example 2



Another Tiny Problem

- Slot machine program
  - Generate random numbers using srand(), time(), rand() functions.
  - There are 3 slots. Each slot may have BAR, BELL, LEMON, CHERRY.
  - The winnings are determined as follows:
  - CHERRY in all 3 slots: JACKPOT
  - CHERRY in any one slot: One Dime
  - same symbols (other than CHERRY) in all 3 slots: NICKEL
  - When the user presses “enter”, run the slot machine program.
  - Expected output:

```
Welcome to KW Land
Please pull the slot machine !!

First is CHERRY
Second is LEMON
Third is LEMON

Paid out : One Dime
Press any key to continue
```

```
Welcome to KW Land
Please pull the slot machine !!

First is BAR
Second is BAR
Third is BAR

Paid out : One Nickel
Press any key to continue
```

Understand the Problem

A slot machine has 3 slots.
Each slot has 4 possible outcomes:
    BAR, BELL, LEMON, CHERRY
Each outcome is random.

3 slots produce outcomes concurrently.

(e.g. BAR LEMON BAR
CHERRY CHERRY BAR
BAR BAR BAR)

winning rules:
    CHERRY in all 3 slots: JACKPOT
    CHERRY in any one slot: DIME
    same outcome in all 3 slots: NICKEL

Outline a Solution

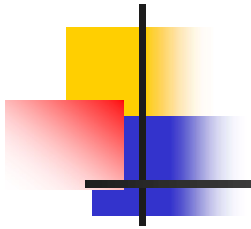
Run the Slot Machine

slot1 generates random outcome
(1 out of 4 possible outcomes).
slot2 generates random outcome
(1 out of 4 possible outcomes).
slot3 generates random outcome
(1 out of 4 possible outcomes).

Determine the Winning

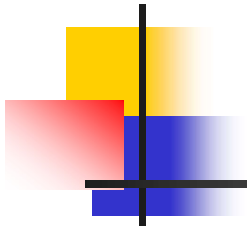
CHERRY in all 3 slots:	JACKPOT
CHERRY in any one slot:	DIME
same outcome in all 3 slots:	NICKEL

Iterate Indefinitely



Flow Graph

- Draw a flow graph as exercise (at two different levels of detail)



Write a Program Outline

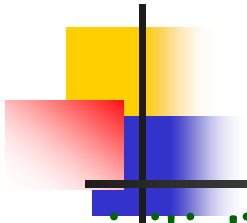
Run the Slot Machine

```
slot1: call rand() (range 1-4)
       assign outcome to slot1
(*  equate 4 outcomes to 4 integers  *)

slot2: call rand() (range 1-4)
       assign outcome to slot2
slot3: call rand() (range 1-4)
       assign outcome to slot3
```

Determine the Winning

```
if slot1=CHERRY and slot2=CHERRY
   and slot3=CHERRY (JACKPOT)
if slot1=CHERRY or slot2=CHERRY or slot3=CHERRY  (DIME)
if slot1=slot2=slot3 and slot1!=CHERRY   (NICKEL)
otherwise   "NO WINNING
```

Iterate Indefinitely



Write a Program Outline

initialize rand()
equate 4 outcomes to 4 integers
declare int slot1, slot2, slot3

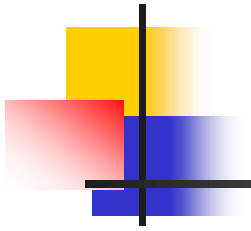
Iterate Indefinitely
    Run the Slot Machine (take any char as input)
        slot1: call rand() (range 1-4)
               assign outcome to slot1
        slot2: call rand() (range 1-4)
               assign outcome to slot2
        slot3: call rand() (range 1-4)
               assign outcome to slot3
    Determine the Winning
        if slot1=CHERRY and slot2=CHERRY
           and slot3=CHERRY (JACKPOT)
        if slot1=CHERRY or slot2=CHERRY or slot3=CHERRY (DIME)
        if slot1=slot2=slot3 and slot1!=CHERRY (NICKEL)
        otherwise "NO WINNING

Code Each Key Component

```
#define BAR 1
#define BELL 2
#define LEMON 3
#define CHERRY 4
#define RMAX 4
initialize rand()

slot1 = 1 + (int) rand() % RMAX;
slot2 = 1 + (int) rand() % RMAX;
slot3 = 1 + (int) rand() % RMAX;

if (slot1==slot2 && slot2==slot3 && slot1==CHERRY)
    printf ("Congratulations On A JACKPOT");
else if (slot1==CHERRY || slot2==CHERRY || slot3==CHERRY)
    printf ("Paid Out: One DIME");
else if (slot1==slot2 && slot2==slot3)
    printf ("Paid Out: One Nickel");
else printf ("Sorry. Better Luck Next Time");
```

Write the Program - 1

```
void main() {

srand(time());
printf ("type any key to start the slot machine");
scanf ("%c", &anykey);

slot1 = 1 + (int) rand() % RMAX;
slot2 = 1 + (int) rand() % RMAX;
slot3 = 1 + (int) rand() % RMAX;

if (slot1==slot2 && slot2==slot3 && slot1==CHERRY)
   printf ("Congratulations On A JACKPOT");
else if (slot1==CHERRY || slot2==CHERRY || slot3==CHERRY)
   printf ("Paid Out: One DIME");
else if (slot1==slot2 && slot2==slot3)
   printf ("Paid Out: One Nickel");
else printf ("Sorry. Better Luck Next Time");
}
```

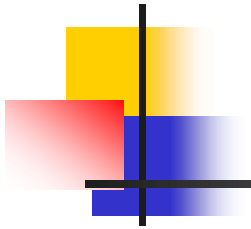


Write the Program – 2 (1/2)

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

#define BAR 1
#define BELL 2
#define LEMON 3
#define CHERRY 4
#define RMAX 4
```

Write the Program - 2 (2/2)

```
void main() {
int slot1, slot2, slot3;
char anykey;
while (1) {
    printf ("type any key to start the slot machine");
    scanf ("%c", &anykey);
    srand(time());
    slot1 = 1 + (int) rand() % RMAX;
    slot2 = 1 + (int) rand() % RMAX;
    slot3 = 1 + (int) rand() % RMAX;
    if (slot1==slot2 && slot2==slot3 && slot1==CHERRY)
        printf ("Congratulations On A JACKPOT");
    else if (slot1==CHERRY || slot2==CHERRY || slot3==CHERRY)
        printf ("Paid Out: One DIME");
    else if (slot1==slot2 && slot2==slot3)
        printf ("Paid Out: One Nickel");
    else printf ("Sorry. Better Luck Next Time");
    }
}
```



End of Class