

**CSE1301**  
**Computer Programming**  
**Lecture 26:**  
**Case Study**

1

**Topics**

- Software quality
- Design principles
- Production principles
- Sample problem: Bingo

2

**Software Quality Features**

- Correctness and Reliability
  - satisfying the **requirements**
- Efficiency
  - obtaining the desired goal with a limited use of **resources** (such as time or hardware)
- Modifiability
  - possibility of adapting software to **changes** in requirements, and to correct errors
- Reusability
  - having modules which may be **useful** in other applications

3

**Design Principles**

- Modular
  - Divide the system into **manageable units**
- Highly cohesive
  - Elements inside a module should be **highly-related** in purpose
- Loosely coupled
  - Maximize **independence** between modules
  - Avoid implicit coupling (such as the use of global variables)

4

**Production Principles**

- Design top-down
  - Break problem down into manageable **sub-problems**
- Build bottom-up
  - Code and test the simplest **components** first
  - Test each component before using them to build more complex components

5

**Production Principles (cont)**

- Develop incrementally
  - Make simplified versions first
  - Add and test one functionality at a time, so that it **evolves** into a complete system
- Document as you go along
  - **System documentation**: describes the software's internal composition
  - **User documentation**: describes how to use the software

6

### Sample Problem: Bingo

I want a program for playing "Bingo"!

Mr. Grouch

The A-Team:

Dave Amy Teddy

7

### Write a Program to Play Bingo

How do we start?

Dave Amy Teddy

8

### Recall: Software Development Cycle

```

    graph LR
      Analysis[Analysis] --> Design[Design]
      Design --> Implement[Implement]
      Implement --> Test[Test]
      Test --> Analysis
      Design --> Analysis
      Implement --> Design
      Test --> Implement
  
```

- Maintain **documentation** of the system throughout the entire cycle

9

### Analysis

- Problem specification:** What are the requirements of the system?

How do you play the game Bingo?  
What are the program's tasks in the game?

10

### The Game of Bingo

- What's involved?
  - Bingo Game Boards
  - A Jar of numbered balls
- Who's involved?
  - The Game Master
  - A Player

11

### Bingo: The Game Board

- A 5 x 5 grid. Each square is a **cell**
- The central cell is **marked** initially

12

### Bingo: The Game Board (cont)

- The boards have **random numbers** assigned to cells in the **ranges** shown (each number can appear *at most once*)

**Range for each column**

6	18	31	58	70
15	23	40	47	71
5	17		51	66
13	29	42	59	62
9	27	34	57	67

1-15
16-30
31-45
46-60
61-75

13

### Bingo: The Jar

- Contains 75 numbered balls

1

2

3

4

5

etc...

14

### Bingo: The Game Master

- At the start of game:
  - Puts all balls into the jar
  - Shakes the jar to mix
- During the game:
  - Takes a randomly-selected ball from the jar
  - Calls out the number on the ball
  - Note:* The balls taken out are *not* returned back to the jar until the next game

15

### Bingo: The Player

- At the start of game:
  - Fills the game board randomly
- During the game:
  - Marks a cell on the game board if it contains the number called out by the Game Master
  - Checks if the board has **winning positions**
  - Calls out "*Bingo!*" if it is a winning board

16

### Bingo: Sample Winning Boards

6	18	31	58	70
15	23	40	47	71
5	17	51	66	
13	29	42	59	62
9	27	34	57	67

6	18		58	
15	23		47	71
	17		51	
13			59	62
9	27		57	67

6		31	58	70
5	17		51	66
	29	42	59	62
9		34	57	

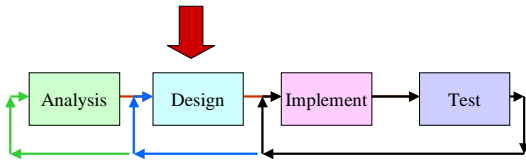
17

### Bingo: End of Game

- The player wins when s/he has a winning game board
- The game ends when
  - there is a winner, or
  - the jar is empty

18

Recall: Software Development Cycle



19

What are the program's tasks?

- Initialization: At the start of the program

```
ask for the name of the player
ask for N (the number of boards for the player)
initialize the player's score to 0
```

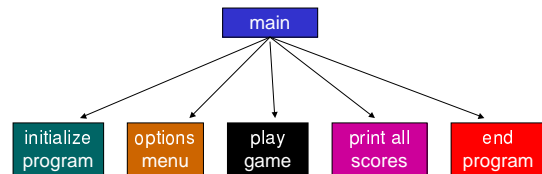
20

What are the program's tasks? (cont)

```
loop
{
  ask what the user wants to do
  if ( user wants to start all over again )
  {
    re-initialize program
  }
  else if ( user wants to play )
  {
    play game
  }
  else if ( user wants to see scores so far )
  {
    print all scores
  }
  else if ( user wants to quit )
  {
    end program /* say goodbye and all that */
  }
}
```

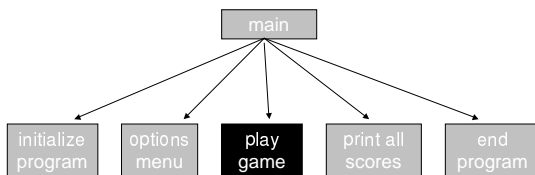
21

Structure Chart (Draft): Main  
(control coupling)



22

Structure Chart (Draft): Main  
(control coupling)



23

Algorithm to Play Bingo

```
place and mix all numbers in jar
fill N game boards randomly
print out all N game boards
while ( player has not won yet )
{
  pick a random number from jar
  update game boards for the player
  print out all N game boards
}
if ( the player won )
{
  congratulate the player
  increment the player's score
}
```

24

### Write a Program to Play Bingo

Help! We're overwhelmed by the task!

Dave Amy Teddy

25

### Write a Program to Play Bingo (cont)

Start off with a simpler version of the problem!

Dave Amy Teddy

26

### Assumptions and Limitations (Version 1)

- Number of boards per player
  - Assume: Only one board per player
- Scoring
  - Assume: No score is kept

27

### Version 1: Algorithm

```

place and mix all numbers in jar
fill game board randomly
print out game board
while ( player has not won yet )
{
    pick a random number from jar
    update player's game board
    print out game board
}
if ( the player won )
{
    congratulate the player
    increment the player's score
}
    
```

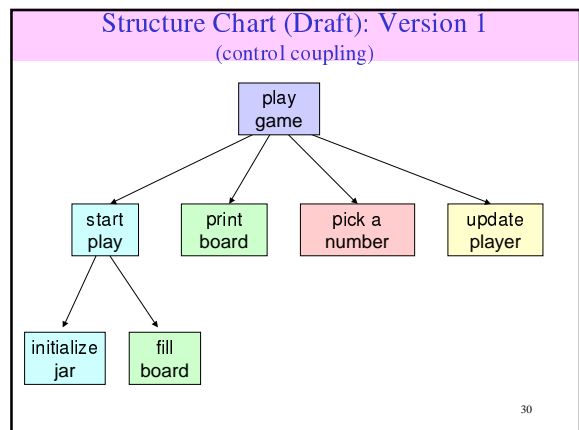
28

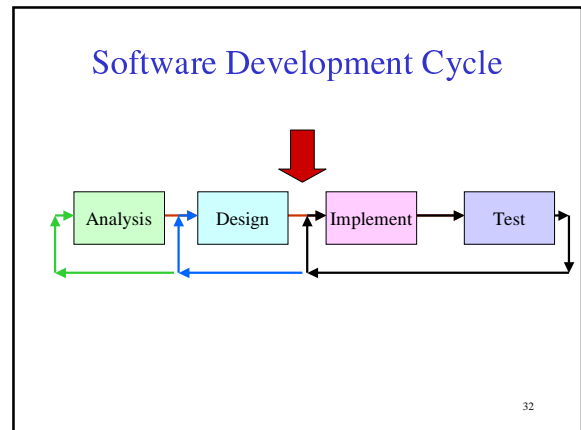
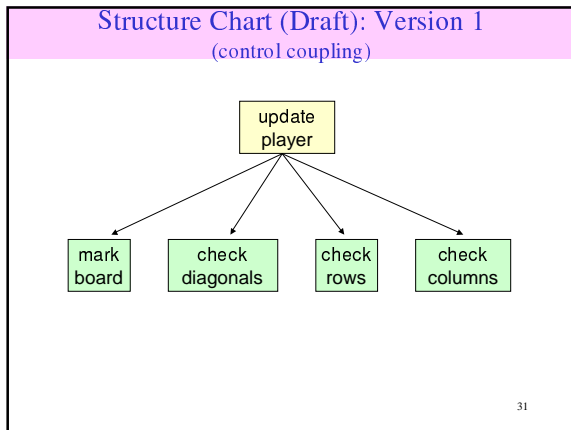
### Version 1: Algorithm (cont)

```

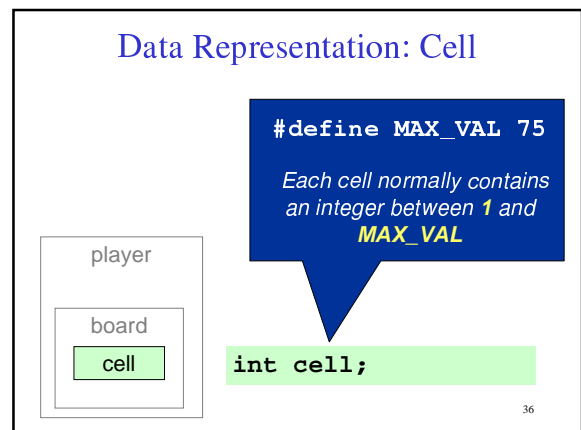
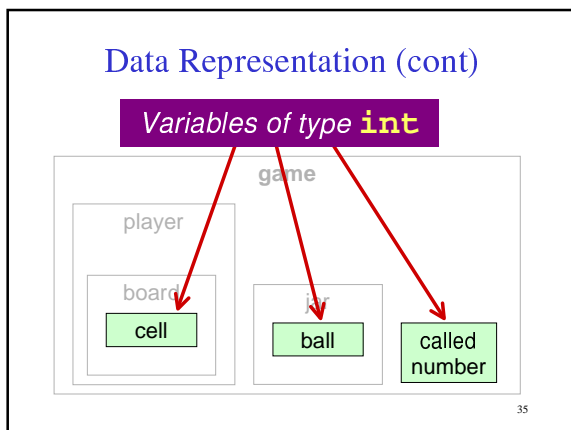
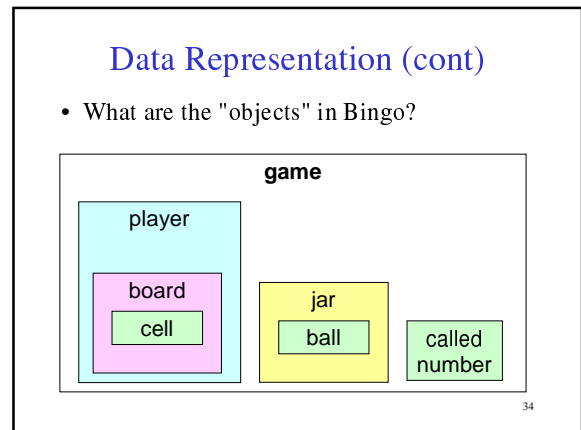
place and mix all numbers in jar
fill game board randomly
print out game board
while ( player has not won yet )
{
    pick a random number from jar
    update player's game board
    print out game board
}
congratulate player
    
```

29





- ### Data Representation
- Identify shared "objects" in the system, and how to **represent** information about them in the program
  - Produce a *data dictionary* of type definitions
    - The type definitions are usually stored in a header file (`bingo.h`), which is shared by all the programs
    - Keep possible future enhancements in mind
- 33



### Data Representation: Cell (cont)

How would you indicate a **marked** cell?

```
#define MARKED_VAL 0
```

A cell with value **MARKED\_VAL** indicates that it has been marked

```
int cell;
```

37

### Data Representation: Board

```
#define BOARD_DIM 5
```

Makes future modifications (large-sized boards) easy

```
struct BingoRec
{
    int cell[BOARD_DIM][BOARD_DIM];
};
typedef struct BingoRec BingoBoard;
```

38

### Data Documentation: Board (cont)

```
/* =====
 *
 * DATA:
 *   BingoBoard
 *
 * DESCRIPTION:
 *   A board is a grid of BOARD_DIM x BOARD_DIM
 *   cells.
 *
 * USAGE:
 *   Each cell normally contains an integer
 *   between 1 and MAX_VAL.
 *   A cell with value MARKED_VAL indicates that
 *   it has been marked.
 *
 * =====
 */
```

39

### Data Representation: Player

```
typedef struct
{
    BingoBoard board;
} PlayerInfo;
```

40

### Data Representation: Player (cont)

```
#define NAME_LEN 80
```

```
typedef struct
{
    BingoBoard board;
    char name[NAME_LEN];
} PlayerInfo;
```

41

### Data Representation: Player (cont)

```
typedef struct
{
    BingoBoard board;
    char name[NAME_LEN];
    int isWinner;
} PlayerInfo;
```

Acts as a "flag": True or false

42

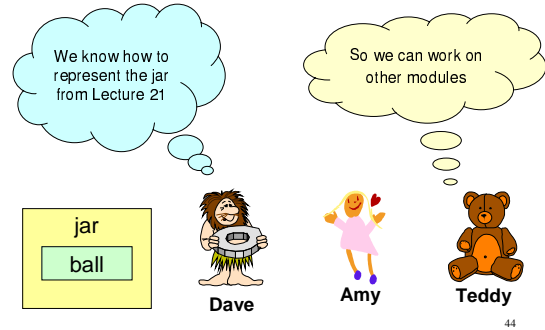
### Data Documentation: Player

```

/* =====
 *
 * DATA:
 *   PlayerInfo
 *
 * DESCRIPTION:
 *   Contains a player's details and bingo board.
 *
 * USAGE:
 *   'isWinner' is 1 if this player currently
 *   has a winning board; 0 otherwise.
 * =====
 */
    
```

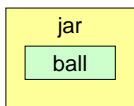
43

### Data Representation: Jar



44

### Data Representation: Jar (cont)



```

typedef struct
{
    int JarArray[MAX_VAL];
    numballs;
} JarInfo;
    
```

45

### Data Representation: Game

```

typedef struct
{
    PlayerInfo    player;
    JarInfo       jar;
    int           calledNumber;
} GameInfo;
    
```

- **Encapsulates** all the information needed for Bingo (*Version 1: one player, one board*)

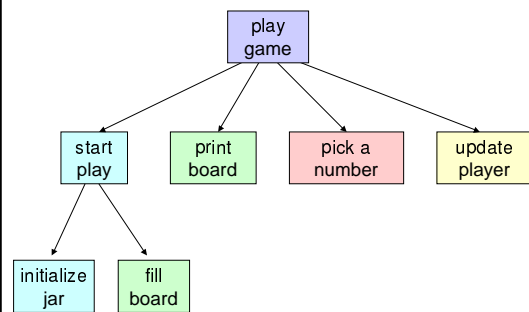
46

### Shared Data

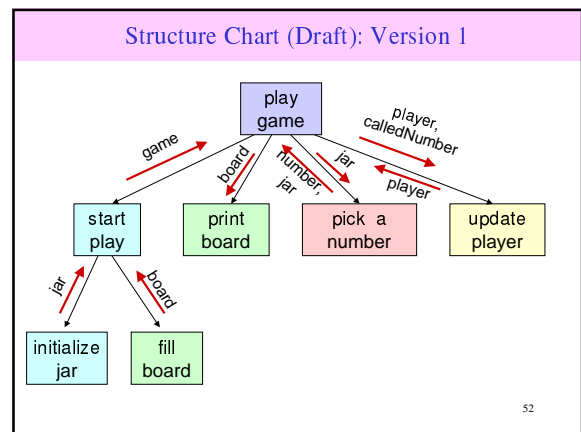
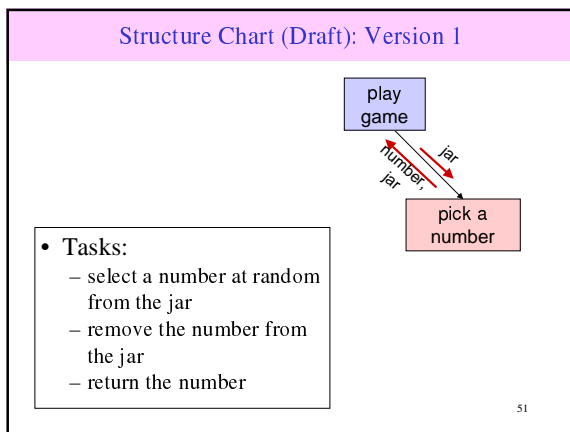
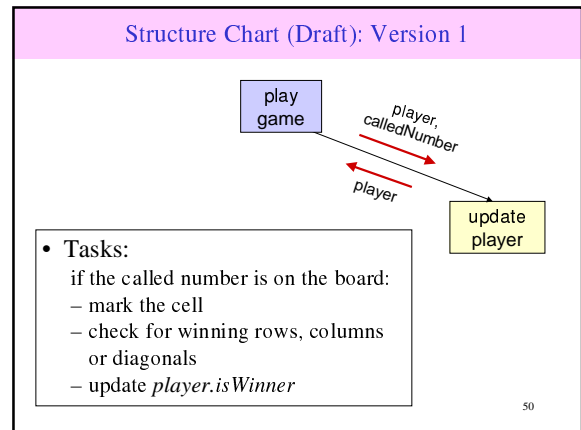
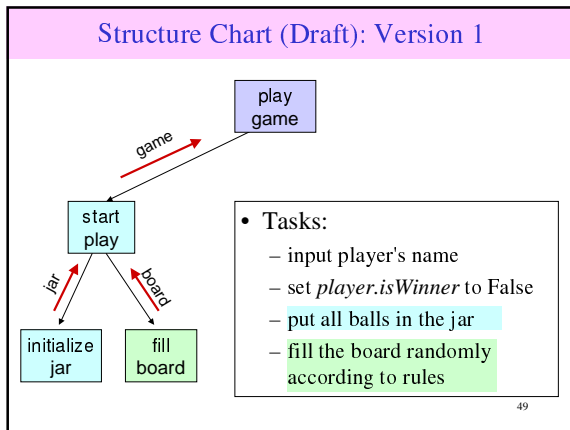
- Information that modules need to share with each other
- **Data coupling** of the modules must be defined
- May require a description of the tasks of each module

47

### Structure Chart (Draft): Version 1



48



### Modular Development

- Recall:
  - Each module (or sub-module) is **implemented** as a C **function**
  - **Execution** of a module (or sub-module) is done with a **function call**
- It is a good idea to store the functions of each component in a separate file

53

### Modular Development (cont)

- Identify "priority" modules
  - Examples:*
    - modules for **initialization** of shared data
      - *startPlay*
    - modules for **testing** other modules
      - *printBoard* -- to test *fillBoard* and *updatePlayer*
    - modules which will help **glue** the sub-modules together
      - *playGame*

54

### Skeleton of Main Module

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

#include "bingo.h" /* data dictionary, #define-s */

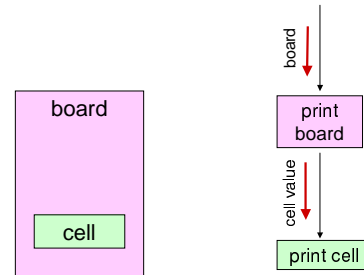
int main()
{
    GameInfo  theGame;

    /* insert code for main algorithm here */

    return 0;
}
```

55

### A "Priority" Module: Print Board



56

### "Stub" for module printBoard

```
/* =====
* NAME:
* void printBoard ( BingoBoard *board )
*
* DESCRIPTION:
* This function is used to print out a bingo board.
* printBoard() calls printCell() to print out an
* individual cell.
* If the cell value is equal to MARKED_VAL, then the cell
* is considered marked. printCell() outputs an indicator
* (such as an asterisk) if the cell is marked.
*
* PRE:
* It is assumed that 'board' points to a struct of type
* BingoBoard, and that the struct has been initialized
* appropriately so that the cells contain only valid
* values.
*
* POST:
* none.
* =====
*/
```

Indicates that the function will **not modify** the board, even if the function uses a pointer

57

bingo-1a-board.c

```
void printCell ( int cellValue )
{
    if ( cellValue == MARKED_VAL )
    {
        printf(" **");
    }
    else if ((1 <= cellValue) && (cellValue <= MAX_VAL))
    {
        printf("%4d", cellValue);
    }
    else
    {
        printf(" BAD");
    }
    return;
}
```

58

### Testing Modules

- A **test program** is used to demonstrate that a module's implementation performs **as expected**
- **Recall: Lecture 15: Flowcharts and debugging**
  - Test data should check every possible execution path of the algorithm

59

### Testing Modules (cont)

- **Build incrementally:** Once a module has been verified using the test program, you can use it to build larger modules
- **Simulate dependencies** to set **controlled conditions** for testing

60

```

#include <stdio.h>
#include <stdlib.h>
#include "bingo.h" /* data dictionary, #define-s */
/* assumes board.c is included in the project */

int main()
{
    /* valid boundary data */
    printCell(1);
    printCell(75);

    /* special case */
    printCell(MARKED_VAL);

    /* invalid data */
    printCell(-1);
    printCell(76);

    printf("\n");
    return 0;
}

```

bingo-1a-test1a.c

61

```

void printBoard ( BingoBoard *board )
{
    int row, col ;

    for ( row = 0; row < BOARD_DIM; row++ )
    {
        printf("\t");
        for ( col = 0; col < BOARD_DIM; col++ )
        {
            printCell ( (*board).cell[row][col] );
        }
        printf("\n");
    }
    return;
}

```

bingo-1a-board.c

62

```

#include <stdio.h>
#include <stdlib.h>
#include "bingo.h" /* data dictionary, #define-s */
/* assumes board.c is included in the project */

const int X = MARKED_VAL;

int main()
{
    /* We'll fill the board up just for testing. */

    BingoBoard theBoard = {{ {1, 16, 31, 46, 61},
                              {2, 17, 32, 47, 62},
                              {3, 18, X, 48, 63},
                              {4, 19, 33, X, 64},
                              {5, 20, 34, 50, 65} }};

    printBoard(&theBoard);

    return 0;
}

```

bingo-1a-test1b.c

63

## System Integration

- Putting **components** together to form larger components, until the entire system is complete
- Often challenging in large systems where several components have to be developed by different people

64

### Code for Main Algorithm -- Version 1

```

#include <stdio.h>
#include <stdlib.h>
#include "bingo.h" /* data dictionary, #define-s */
/* assumes board.c, jar.c, player.c are included */
int main()
{
    GameInfo theGame;

    startPlay(&theGame);
    printBoard(&(theGame.player.board));

    while (!(theGame.player.isWinner))
    {
        callOutNumber(&(theGame.jar),
                     &(theGame.calledNumber));
        updatePlayer(&(theGame.player));
    }

    printf("BINGO! Congrats, %s!\n", theGame.player.name);
    return 0;
}

```

65

### On to Bingo -- Version 2

- Several boards per player
- Keep the scores in a continuing game

66

## Reading

- Brookshear (5/e or 6/e): Chapter 6

67