



PROGRAMMING METHODOLOGY (PHƯƠNG PHÁP LẬP TRÌNH)

UNIT 5: Selection Statements

Acknowledgement

- The contents of these slides have origin from School of Computing, National University of Singapore.
- We greatly appreciate support from Mr. Aaron Tan Tuck Choy for kindly sharing these materials.

Policies for students

- These contents are only used for students PERSONALLY.
- Students are NOT allowed to modify or deliver these contents to anywhere or anyone for any purpose.

Recording of modifications

- Currently, there are no modification on these contents.

Unit 5: Selection Statements

Objectives:

- Using relational and logical operators
- Using selection statements to choose between two or more execution paths in a program
- Formulating complex selection structures to solve decision problems

Reference:

- Chapter 4 Lessons 4.1 – 4.6, Beginning Decision Making

Unit 5: Selection Statements (1/2)

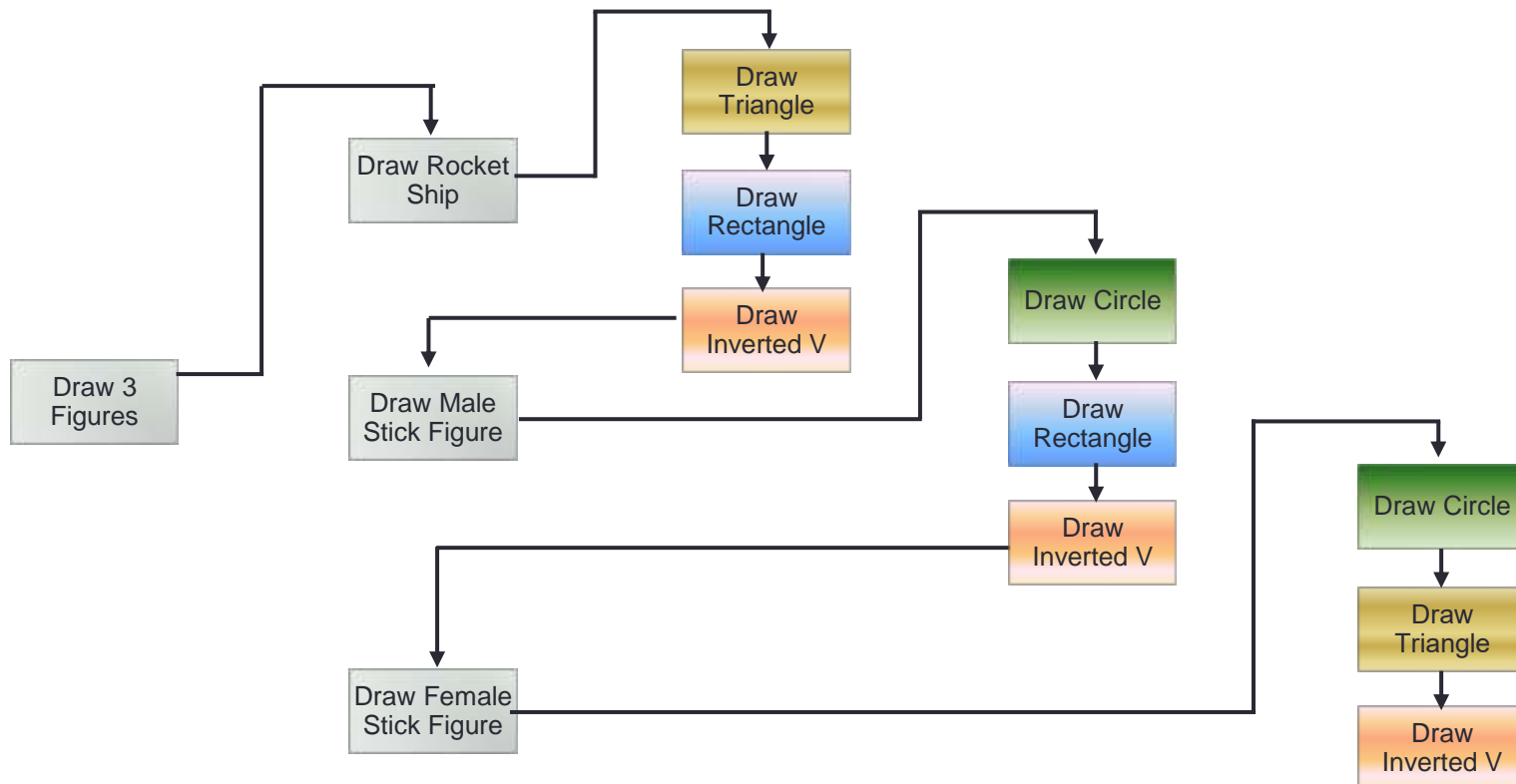
1. Sequential vs Non-Sequential Control Flow
2. Selection Structures
 - 2.1 *if* and *if-else* Statements
 - 2.2 Conditions
 - 2.3 Truth Values
 - 2.4 Logical Operators
 - 2.5 Evaluation of Boolean Expressions
 - 2.6 Caution
 - 2.7 Short-Circuit Evaluation
 - 2.8 *if* and *if-else* Statements: Examples

Unit 5: Selection Statements (2/2)

3. Nested *if* and *if-else* Statements
4. Style Issues
5. Common Errors
6. The *switch* Statement
7. Testing and Debugging

1. Sequential Control Flow

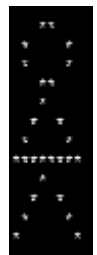
- Recall Simple “drawing” problem in Unit 4:
Write a program to draw a rocket ship, a male stick figure, and a female stick figure.



rocket



male

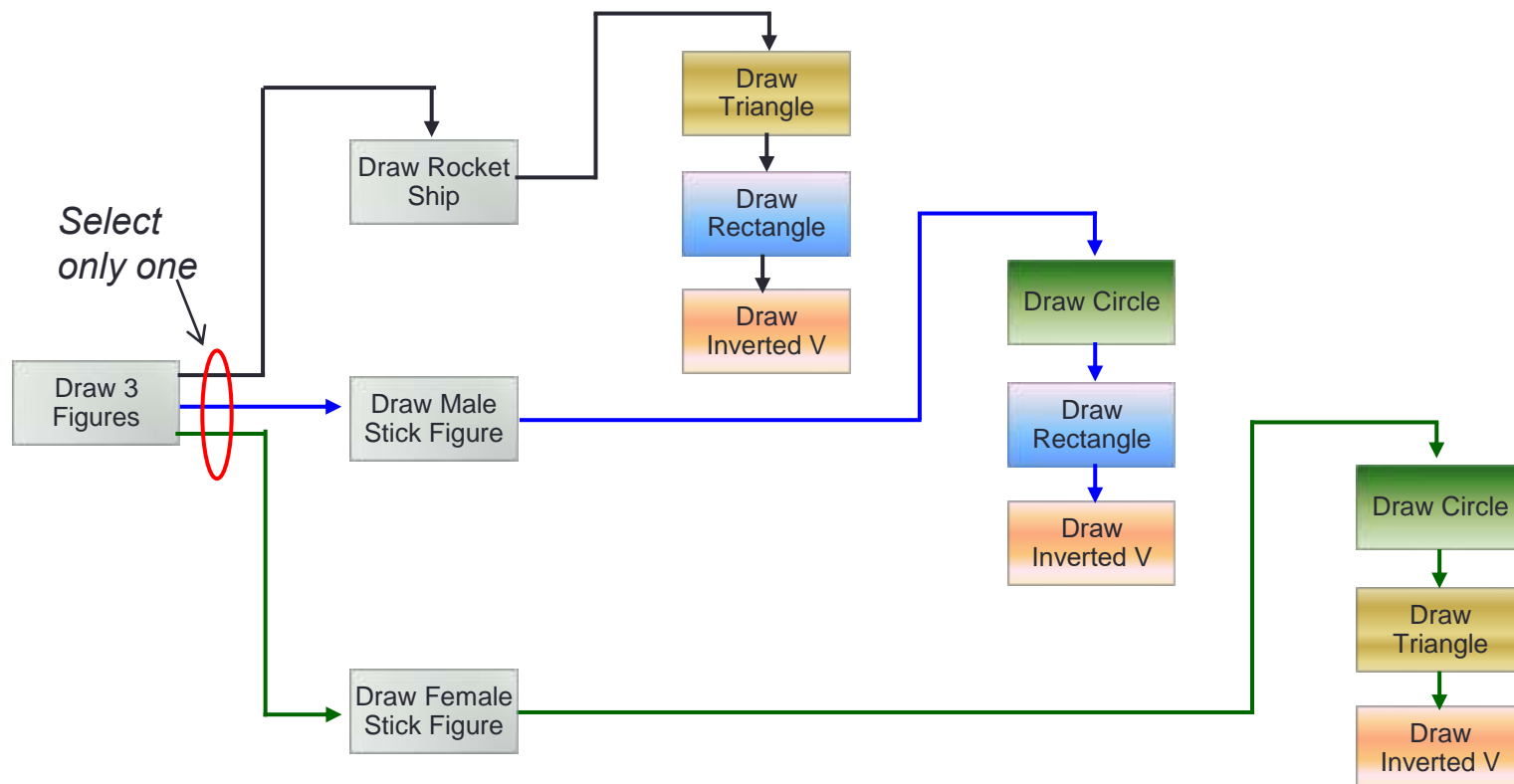


female

1. Non-Sequential Control Flow

- New requirement:

Write a program to allow user to select only ONE of the following options: Draw a (1) rocket ship, (2) male stick figure, or (3) female stick figure.



2. Selection Structures

- C provides two control structures that allow you to select a group of statements to be executed or skipped when certain conditions are met.

if ... else ...

switch

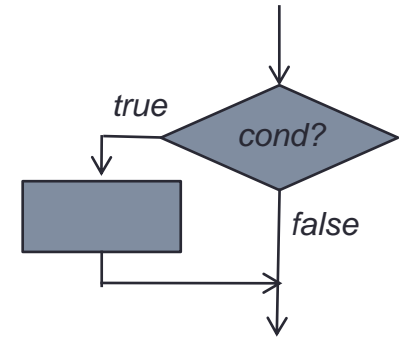


2.1 *if* and *if-else* Statements

- *if* statement

How are conditions specified and how are they evaluated?

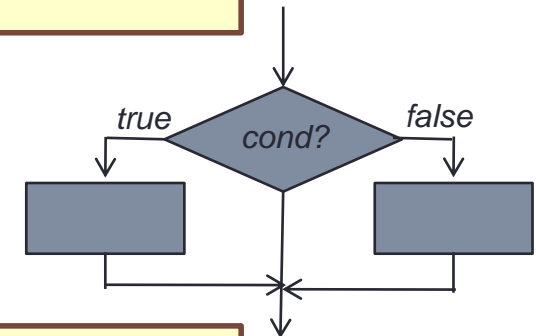
```
if ( condition ) {  
    /* Execute these statements if TRUE */  
}
```



- *if-else* statement

Braces { } are optional only if there is one statement in the block.

```
if ( condition ) {  
    /* Execute these statements if TRUE */  
}  
else {  
    /* Execute these statements if FALSE */  
}
```



2.2 Condition

- A **condition** is an expression evaluated to true or false.
- It is composed of expressions combined with **relational operators**.
 - Examples: $(a \leq 10)$, $(\text{count} > \text{max})$, $(\text{value} \neq -9)$

| Relational Operator | Interpretation |
|---------------------|-----------------------------|
| < | is less than |
| <= | is less than or equal to |
| > | is greater than |
| >= | is greater than or equal to |
| == | is equal to |
| != | is not equal to |

2.3 Truth Values

- Boolean values: **true** or **false**.
- There is no boolean type in ANSI C. Instead, we use integers:
 - **0** to represent **false**
 - **Any other value** to represent **true** (**1** is used as the representative value for true in output)
- Example:

Unit5_TruthValues.c

```
int a = (2 > 3);  
int b = (3 > 2);
```

```
printf("a = %d; b = %d\n", a, b);
```

a = 0; b = 1

2.4 Logical Operators

- **Complex condition**: combining two or more boolean expressions.
- Examples:
 - If temperature is greater than 40C **or** blood pressure is greater than 200, go to A&E immediately.
 - If all the three subject scores (English, Maths **and** Science) are greater than 85 **and** mother tongue score is at least 80, recommend taking Higher Mother Tongue.
- **Logical operators** are needed: **&&** (and), **||** (or), **!** (not).

| A | B | A && B | A B | !A |
|-------|-------|--------|--------|-------|
| False | False | False | False | True |
| False | True | False | True | True |
| True | False | False | True | False |
| True | True | True | True | False |

Note: There are **bitwise operators** such as **&**, **|** and **^**, but we are not covering these in CS1010.

2.5 Evaluation of Boolean Expressions (1/2)

- The evaluation of a boolean expression is done according to the **precedence** and **associativity** of the operators.

| Operator Type | Operator | Associativity |
|------------------------------|---|---------------|
| Primary expression operators | () [] . -> expr++ expr-- | Left to Right |
| Unary operators | * & + - ! ~ ++expr --expr (typecast) sizeof | Right to Left |
| Binary operators | * / % | Left to Right |
| | + - | |
| | < > <= >= | |
| | == != | |
| | && | |
| | | |
| Ternary operator | ?: | Right to Left |
| Assignment operators | = += -= *= /= %= | Right to Left |

2.5 Evaluation of Boolean Expressions (2/2)

See Unit5_EvalBoolean.c

- What is the value of **x**?

```
int x, y, z,  
    a = 4, b = -2, c = 0;  
x = (a > b || b > c && a == b);
```

x is true (1)

gcc issues warning (why?)

- Always good to add parentheses for readability.

```
y = ((a > b || b > c) && a == b);
```

y is false (0)

- What is the value of **z**?

```
z = ((a > b) && !(b > c));
```

z is true (1)

2.6 Caution (1/2)



- Since the values 0 and 1 are the returned values for **false** and **true** respectively, we can have codes like these:

```
int a = 12 + (5 >= 2); // 13 is assigned to a
```

(5 >= 2) evaluates to 1; hence a = 12 + 1;

```
int b = (4 > 5) < (3 > 2) * 6; // 1 assigned to b
```

* has higher precedence than <.

(3 > 2) evaluates to 1, hence (3 > 2) * 6 evaluates to 6.

(4 > 5) evaluates to 0, hence 0 < 6 evaluates to 1.

```
int c = ((4 > 5) < (3 > 2)) * 6; // 6 assigned to c
```

(4 > 5) evaluates to 0, (3 > 2) evaluates to 1, hence

(4 > 5) < (3 > 2) is equivalent to (0 < 1) which evaluates to 1.

Hence 1 * 6 evaluates to 6.

- You are certainly not encouraged to write such convoluted codes!

2.6 Caution (2/2)



- **Very** common mistake:

```
int num;

printf("Enter an integer: ");
scanf("%d", &num);

if (num = 3) {
    printf("The value is 3.\n");
}
printf("num = %d\n", num);
```

- What if user enters 7?
- Correct the error.

2.7 Short-Circuit Evaluation

- Does the following code give an error if variable **a** is zero?

```
if ((a != 0) && (b/a > 3))  
    printf(. . .);
```

- Short-circuit evaluation

- **expr1 || expr2**: If expr1 is true, skip evaluating expr2 and return true immediately, as the result will always be true.
- **expr1 && expr2**: If expr1 is false, skip evaluating expr2 and return false immediately, as the result will always be false.

2.8 *if* and *if-else* Statements: Examples (1/2)

if statement
without *else* part

```
int a, b, t;
. . .
if (a > b) {
    // Swap a with b
    t = a; a = b; b = t;
}
// After above, a is the smaller
```

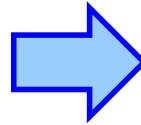
if-else statement

```
int a;
. . .
if (a % 2 == 0) {
    printf("%d is even\n", a);
}
else {
    printf("%d is odd\n", a);
}
```

2.8 *if* and *if-else* Statements: Examples (2/2)

- Move common statements out of the *if-else* construct.

```
if (cond) {  
    statement-a;  
    statement-b;  
    statement-j;  
    statement-x;  
    statement-y;  
}  
else {  
    statement-a;  
    statement-b;  
    statement-k;  
    statement-x;  
    statement-y;  
}
```



```
statement-a;  
statement-b;  
if (cond) {  
    statement-j;  
}  
else {  
    statement-k;  
}  
statement-x;  
statement-y;
```

3. Nested *if* and *if-else* Statements (1/2)

- Nested *if* (*if-else*) structures refer to the containment of an *if* (*if-else*) structure within another *if* (*if-else*) structure.
- For example:
 - If it is a weekday, you will be in school from 8 am to 6 pm, do revision from 6 pm to 12 midnight, and sleep from 12 midnight to 8 am.
 - If it is a weekend, then you will sleep from 12 midnight to 10 am and have fun from 10 am to 12 midnight.

3. Nested *if* and *if-else* Statements (2/2)

- Drawing task in Unit 4

```
int main(void) {  
    draw_rocket();  
    printf("\n\n");  
    draw_male();  
    printf("\n\n");  
    draw_female();  
    printf("\n\n");  
  
    return 0;  
}
```

- Draw only 1 figure

```
int main(void) {  
    char resp;  
  
    printf(" (R)ocket, ");  
    printf(" (M)ale, or ");  
    printf(" (F)emale? ");  
    scanf("%c", &resp);  
  
    if (resp == 'R')  
        draw_rocket();  
    else if (resp == 'M')  
        draw_male();  
    else if (resp == 'F')  
        draw_female();  
  
    return 0;  
}
```

4. Style Issues: Indentation (1/6)

- Once we write non-sequential control structures, we need to pay attention to indentation.

Acceptable

```
if (cond) {  
    statements;  
}  
else {  
    statements;  
}
```

```
if (cond) {  
    statements;  
} else {  
    statements;  
}
```

```
if (cond)  
{  
    statements;  
}  
else  
{  
    statements;  
}
```

Do you
remember which
vim command to
auto-indent your
program?

Non-acceptable

```
if (cond)  
{  
    statements;  
}  
else  
{  
    statements;  
}
```

No indentation!

```
if (cond) {  
    statements;}  
else {  
    statements;}
```

*Closing braces
not aligned with
if/else keyword!*

4. Style Issues: Indentation (2/6)

- Note that appropriate indentation of comments is just as important.

Correct

```
// Comment on the whole if
// construct should be aligned with
// the 'if' keyword
if (cond) {
    // Comment on the statements in
    // this block should be aligned
    // with the statements below
    statements;
}
else {
    // Likewise, comment for this
    // block should be indented
    // like this
    statements;
}
```

Incorrect

```
    // Compute the fare
if (cond) {
    // For peak hours
    statements;
}
else {
    // For non-peak hours
    statements;
}
```

4. Style Issues: Indentation (3/6)

- Sometimes we may have a deeply nested *if-else-if* construct:

```
int marks;  
char grade;  
.  
.  
.  
if (marks >= 90)  
    grade = 'A';  
else  
    if (marks >= 75)  
        grade = 'B';  
    else  
        if (marks >= 60)  
            grade = 'C';  
        else  
            if (marks >= 50)  
                grade = 'D';  
            else  
                grade = 'F';
```

- This follows the indentation guideline, but in this case the code tends to be long and it skews too much to the right.

4. Style Issues: Indentation (4/6)

- Alternative (and preferred) indentation style for deeply nested *if-else-if* construct:

```
int marks;  
char grade;  
.  
.  
.  
if (marks >= 90)  
    grade = 'A';  
else  
    if (marks >= 75)  
        grade = 'B';  
    else  
        if (marks >= 60)  
            grade = 'C';  
        else  
            if (marks >= 50)  
                grade = 'D';  
            else  
                grade = 'F';
```

Alternative style

```
int marks;  
char grade;  
.  
.  
.  
if (marks >= 90)  
    grade = 'A';  
else if (marks >= 75)  
    grade = 'B';  
else if (marks >= 60)  
    grade = 'C';  
else if (marks >= 50)  
    grade = 'D';  
else  
    grade = 'F';
```

4. Style Issues: Naming 'boolean' variables (5/6)

- Here, 'boolean' variables refer to `int` variables which are used to hold 1 or 0 to represent true or false respectively.
- These are also known as `boolean flags`.
- To improve readability, boolean flags should be given descriptive names just like any other variables.
- In general, add suffices such as "is" or "has" to names of boolean flags (instead of just calling them "flag"!)
 - Example: `isEven`, `isPrime`, `hasError`, `hasDuplicates`

```
int isEven, num;  
.  
.  
.  
if (num % 2 == 0)  
    isEven = 1;  
else  
    isEven = 0;
```

4. Style Issues: Removing 'if' (6/6)

- The following code pattern is commonly encountered:

```
int isEven, num;  
.  
.  
.  
if (num % 2 == 0)  
    isEven = 1;  
else  
    isEven = 0;
```

- In this case, the *if* statement can be rewritten into a single assignment statement, since $(\text{num} \% 2 == 0)$ evaluates to either 0 or 1.
- Such coding style is common and the code is shorter.

```
int isEven, num;  
.  
.  
.  
isEven = (num % 2 == 0);
```

5. Common Errors (1/2)

- The code fragments below contain some very common errors. One is caught by the compiler but the other is not (which makes it very hard to detect). **Spot the errors.**

Unit5_CommonErrors1.c

```
int a = 3;  
if (a > 10);  
    printf("a is larger than 10\n");  
printf("Next line.\n");
```

Unit5_CommonErrors2.c

```
int a = 3;  
if (a > 10);  
    printf("a is larger than 10\n");  
else  
    printf("a is not larger than 10\n");  
printf("Next line.\n");
```

5. Common Errors (2/2)

- Proper indentation is important. In the following code, the indentation does not convey the intended purpose of the code. Why? Which *if* is the *else* matched to?

Unit5_CommonErrors3.c

```
int a, b;  
.  
.  
.  
  
if (a > 10)  
    if (b < 9)  
        printf("Hello\n");  
else  
    printf("Goodbye\n");
```

6. The *switch* Statement (1/3)

- An alternative to *if-else-if* is to use the *switch* statement.
- Restriction: Value must be of **discrete type** (eg: *int*, *char*)

```
switch ( <variable or expression> ) {  
    case value1:  
        Code to execute if <variable or expr> == value1  
        break;  
  
    case value2:  
        Code to execute if <variable or expr> == value2  
        break;  
    ...  
  
    default:  
        Code to execute if <variable or expr> does not  
        equal to the value of any of the cases above  
        break;  
}
```


6. The *switch* Statement (2/3)

- Write a program that reads in a **6-digit zip code** and uses its first digit to print the associated geographic area.

| If zip code begins with | Print this message |
|-------------------------|--------------------------------------|
| 0, 2 or 3 | <zip code> is on the East Coast. |
| 4 – 6 | <zip code> is in the Central Plains. |
| 7 | <zip code> is in the South. |
| 8 or 9 | <zip code> is in the West. |
| others | <zip code> is invalid. |

6. The *switch* Statement (3/3)

Unit5_ZipCode.c

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

    printf("Enter a 6-digit ZIP code: ");
    scanf("%d", &zip);

    switch (zip/100000) {
        case 0: case 2: case 3:
            printf("%06d is on the East Coast.\n", zip);
            break;
        case 4: case 5: case 6:
            printf("%d is in the Central Plains.\n", zip);
            break;
        case 7:
            printf("%d is in the South.\n", zip);
            break;
        case 8: case 9:
            printf("%d is in the West.\n", zip);
            break;
        default:
            printf("%d is invalid.\n", zip);
    } // end switch

    return 0;
}
```

7. Testing and Debugging (1/3)

- Finding the maximum value among 3 variables:

```
// Returns largest among num1, num2, num3
int getMax(int num1, int num2, int num3) {
    int max = 0;
    if ((num1 > num2) && (num1 > num3))
        max = num1;
    if ((num2 > num1) && (num2 > num3))
        max = num2;
    if ((num3 > num1) && (num3 > num2))
        max = num3;
    return max;
}
```

Unit5_FindMax_v1.c

- What is wrong with the code? Did you test it with the correct test data?
- What test data would expose the flaw of the code?
- How do you correct the code?
- After correcting the code, would replacing the 3 *if* statements with a nested *if-else* statement work? If it works, which method is better?

7. Testing and Debugging (2/3)

- With selection structures (and next time, repetition structures), you are now open to many alternative ways of solving a problem.
- Alternative approach to finding maximum among 3 values:

```
// Returns largest among num1, num2, num3
int getMax(int num1, int num2, int num3) {
    int max = 0;
    if (num1 > max)
        max = num1;
    else if (num2 > max)
        max = num2;
    else if (num3 > max)
        max = num3;
    return max;
}
```

Unit5_FindMax_v2.c

- What is wrong with this code? (There are more than one error.)
- What test data should you use to expose its flaw?

7. Testing and Debugging (3/3)

- The preceding examples will be discussed in class.
- Remember: Test your programs thoroughly with your own data.

Do NOT rely on
CodeCrunch to test your
programs!

Summary

- In this unit, you have learned about
 - The use of *if-else* construct and *switch* construct to alter program flow
 - The use of relational and logical operators
 - Style issues such as indentation, naming of boolean flags and replacing *if* statement with an assignment statement
 - How to test a selection construct with exhaustive test data, and to ensure that all alternative paths in the selection construct are examined

End of File