Logic and Selection 2

Session 3 12 November 2002
An adult badminton training class organized by the Cultural and Recreation Department welcomes anyone aged at least 18 and below 65 to join (there are youth and senior classes for other age groups). The following program is written for the purpose of checking the age requirement.

The program follows the same structure as the programs in the previous Perform Exercises. Make sure you can identify the program header (the `#include` and `void main` part), the variable declaration part, the input part, the processing part, and the output part.
The above program prints, "Your age satisfies ..." if age is greater than or equal to 18 and the age is less than 65. It prints "Sorry. Your ..." if age is any other value.
Achieve Target 1

Rewrite the below `if-else` structure so that the `if` condition tests for age outside the required range. Work on the program below to make it happen.

```c
... if ( /* rewrite the condition */ ) {
    printf("Sorry. Your age does not satisfy the requirement\n");
}
else {
    printf("Your age satisfies the requirement\n");
}
...
```c
#include <stdio.h>

void main() {
    int age;
    printf("Enter your age: ");
    scanf("%d", &age);

    if (age < 18 || age >= 65) {
        printf("Sorry. Your age does not satisfy the requirement\n");
    } else {
        printf("Your age satisfies the requirement\n");
    }

    fflush(stdin);
    getchar();
}
```
Achieve Target 1

The solution involves you to think of the opposite condition. If the required age is greater than or equal to 18 and less than 65, then the age not satisfying the requirement is less than 18 or greater or equal to 65.

```c
... if (age < 18 || age >= 65) {
    printf("Sorry.  Your age does not satisfy the requirement\n");
}
else {
    printf("Your age satisfies the requirement\n");
}
...```
Achieve Target 1A

A bus company offers half fare for those aged below 12 and those aged on or above 65. The following program is used for calculating the correct fare to tender.

```c
#include <stdio.h>

void main() {

    int age;
    float fare;

    printf("Enter your age: ");
    scanf("%d", &age);
    printf("Enter the bus fare: ");
    scanf("%f", &fare);

    if (age < 12 || age >= 65)
        fare = fare / 2; /* pay half fare */

    printf("The fare to tender is $%f\n", fare);
    fflush(stdin);
    getchar();
}
```

The program uses variables of both **int** and **float** types.

Note that the uses of "%d" and "%f" match the type of the variable.

The "%d" is used with **int** variables/values and "%f" is used with **float** variables/values.
Now the bus company has finally succumbed to public pressure and agreed to offer a **10%** fare discount to everybody, *except* those already on **half fare**.

```c
evaluate <stdio.h>
void main() {
    int age;
    float fare;

    printf("Enter your age: ");
    scanf("%d", &age);
    printf("Enter the bus fare: ");
    scanf("%f", &fare);

    if (age < 12 || age >= 65)
        fare = fare / 2; /* pay half fare */

    printf("The fare to tender is $\%f\n", fare);
    fflush(stdin);
    getchar();
}
```

**Work** on the above program to make it happen.

*Ask yourself what is the condition to apply **10%** discount, and what is the condition to apply **50%** discount (half fare).*
#include <stdio.h>

void main() {

    int age;
    float fare;

    printf("Enter your age: ");
    scanf("%d", &age);
    printf("Enter the bus fare: ");
    scanf("%f", &fare);

    if (age < 12 || age >= 65)
        fare = fare / 2;  /* pay half fare */
    else
        fare = fare * 0.9;  /* a 10% discount */

    printf("The fare to tender is $%f\n", fare);

    fflush(stdin);
    getchar();
}
The following is a program segment extracted from the BMI Health Evaluator program in the previous Perform Exercise.

```c
...printf("BMI = %f\n", bmi);

if (bmi < 19)
    printf("Underweight\n");
if (bmi >= 19 && bmi <= 22)
    printf("Just Right\n");
if (bmi > 22 && bmi <= 25)
    printf("Overweight\n");
if (bmi > 25)
    printf("Health at Risk\n");
...```

Each operator requires the computer's effort to execute.
Achieve Target 2

Count how many operators are executed in the processing below.

```c
... printf("BMI = %f\n", bmi);

if (bmi < 19)
    printf("Underweight\n");
if (bmi >= 19 && bmi <= 22)
    printf("Just Right\n");
if (bmi > 22 && bmi <= 25)
    printf("Overweight\n");
if (bmi > 25)
    printf("Health at Risk\n");
...```

Does the value of bmi affect the count? Work on the problem.
Solution

We simply count the number of operators involved in the four if structures in the first program. We would get 8 (1 for the first if structure, 3 for the second and third, and 1 for the last).

```c
printf("BMI = %f\n", bmi);

if (bmi < 19)
    printf("Underweight\n");
if (bmi >= 19 && bmi <= 22)
    printf("Just Right\n");
if (bmi > 22 && bmi <= 25)
    printf("Overweight\n");
if (bmi > 25)
    printf("Health at Risk\n");
```
Achieve Target 2A

The program is now rewritten using \texttt{if-else} structures.

\begin{verbatim}
... 
printf("BMI = %f\n", bmi);

    if (bmi < 19)
        printf("Underweight\n");
    else if (bmi >= 19 && bmi <= 22)
        printf("Just Right\n");
    else if (bmi > 22 && bmi <= 25)
        printf("Overweight\n");
    else
        printf("Health at Risk\n");
...
\end{verbatim}

Note that at the last \texttt{else} part no \texttt{if} condition is required. For the \texttt{statement in blue} to execute, all of the three \texttt{if} conditions must evaluate to \texttt{zero}.
Now count how many operators are executed in the processing below.

```c
...
printf("BMI = %f\n", bmi);

if (bmi < 19)
    printf("Underweight\n");
else if (bmi >= 19 && bmi <= 22)
    printf("Just Right\n");
else if (bmi > 22 && bmi <= 25)
    printf("Overweight\n");
else
    printf("Health at Risk\n");
...
```

Does the value of bmi affect the count?
What would be the minimum and the maximum possible number of operators executed? Work on the problem.
A careful use of *if–else* changes the count significantly.

```c
... printf("BMI = %f\n", bmi);

if (bmi < 19)
    printf("Underweight\n");
else if (bmi >= 19 && bmi <= 22)
    printf("Just Right\n");
else if (bmi > 22 && bmi <= 25)
    printf("Overweight\n");
else
    printf("Health at Risk\n");

...```

The minimal number of operators executed is 1. If the value of variable `bmi` is less than 19, then only the first *if* statement is executed and the rest would be ignored.

The maximum number of operators executed is 7, in the cases when the variable `bmi` is greater than 25.
Achieve Target 2A

The observation is that in general *if-else* structure can reduce the execution of a program, therefore making a program faster with less amount of execution.

Please note that the numbers in the previous program are not absolutely correct in reality. The reason will be explained later in the course. Anyway, we are interested in their relative counts only.
Achieve Target 2B

Remember the program in the previous Perform Exercise that calculates the number of free cans to give out in a soft-drink promotion.

```c
#include <stdio.h>

void main() {
    int numberOfCan;
    int freeCan;

    printf("Enter the number of cans purchased: ");
    scanf("%d", &numberOfCan);

    freeCan = numberOfCan / 5;
    if (numberOfCan >= 30) {
        freeCan += 3;
    }
    printf("The customer should receive extra %d cans free\n", freeCan);

    ...
```

One free can is given for every 5 cans purchased, and 3 extra cans are given if the number of cans purchased is at or over 30.
Now the promotion scheme has changed. Modify the processing part in blue so that the following promotion schemes applies. Use if-else structures in your work. Work on the program to make it happen.

```c
...  freeCan = numberOfCan / 5;
  if (numberOfCan >= 30) {
    freeCan += 3;
  }
  printf("The customer should receive extra %d cans
         free\n", freeCan);
...
```

Two free cans are given if cans purchased is between 20 and 40 inclusive.

Three free cans are given if cans purchased is between 41 and 60.

Four free cans are given if over 60 are purchased.

(So 2 free cans for 40, 3 free cans for 41, and 4 free cans for 61)
Achieve Target 2B

Solution 1

if (numberOfCan >= 20 && numberOfCan <= 40) {
    freeCan = 2;
} else if (numberOfCan >= 41 && numberOfCan <= 60) {
    freeCan = 3;
} else if (numberOfCan > 60) {
    freeCan = 4;
} else {
    freeCan = 0;
}
Solution 2

An alternative method that uses no if-else structure

```plaintext
... freeCan = (numberOfCan >= 20 && numberOfCan <= 40) * 2 +
          (numberOfCan >= 41 && numberOfCan <= 60) * 3 +
          (numberOfCan > 60) * 4;
... 
```
Promotion Scheme 2

One free cans are given for every 8 cans.

Four extra cans are given if cans purchased is at least 30.

An additional four extra cans are given if cans purchased is over 50.

\[
\text{freeCan} = \frac{\text{numberOfCan}}{5};
\text{if (numberOfCan} \geq 30) \{ \\
\text{freeCan} += 3; \\
\text{\}
\]

\text{printf("The customer should receive extra %d cans free\n", freeCan);}

\text{(So 3 free cans for 29, 7 free cans for 30, and 14 free cans for 51)}
Achieve Target 2C

Solution

```java
... freeCan = numberOfCan / 8;
    if (numberOfCan >= 30)
        freeCan = freeCan + 4;
    if (numberOfCan > 50)
        freeCan = freeCan + 4;
... 

... freeCan = (numberOfCan / 8) +
            (numberOfCan >= 30) * 4 +
            (numberOfCan > 50) * 4;
... 
```
Achieve Target 2D

Promotion Scheme 3
For every 5 cans purchased over 20, one free can is given.
For every 5 cans purchased over 50, two free cans is given instead of one.
A customer can get at most 10 free cans.

```c
... freeCan = numberOfCan / 5;
if (numberOfCan >= 30) {
  freeCan += 3;
}
printf("The customer should receive extra %d cans free\n", freeCan);
...
```

So 0 free cans for 20, 0 free cans for 24, and 1 free cans for 25, 10 free cans for 60
A separate if statement is needed for the "at most 10 free cans" rule.
Achieve Target 2D

Solution

... 

if (numberOfCan > 20 && numberOfCan <= 50) {
    freeCan = (numberOfCan - 20) / 5;
}
else if (numberOfCan > 50) {
    freeCan = 6 + (numberOfCan - 50) * 2 / 5;
}
if (freeCan > 10)
    freeCan = 10;

...
Achieve Target 2D

Note that there is the second if structure there to check for the condition where `freeCan` is greater than 10.

```java
... if (numberOfCan > 20 && numberOfCan <= 50) {
    freeCan = (numberOfCan - 20) / 5;
} else if (numberOfCan > 50) {
    freeCan = 6 + (numberOfCan - 50) * 2 / 5;
}
if (freeCan > 10)
    freeCan = 10;
...```

Remember it.

By counting the number of operator execution required, we can easily tell that the if-else structure would execute fewer operators and therefore slightly faster.
A department store offers a membership scheme so that members get discount off any goods purchased. Three levels of membership (Gold, Silver and Bronze) offer different discount rate at different purchase amount.

<table>
<thead>
<tr>
<th>Level</th>
<th>Discount Details</th>
</tr>
</thead>
</table>
| Gold   | 20% discount for purchase amount at least $500 and less than $1000  
         | 30% discount for purchase amount over $1000 |
| Silver | 20% discount for purchase amount at least $1000       |
| Bronze | 10% discount for purchase amount at least $2000       |
We now uses a new type of variable, the `char` type.

The `%c` is used in `scanf` and `printf` for `char` type variables/values.

```
#include <stdio.h>
void main() {
    char type;
    float amount;

    printf("Enter membership type\n");
    printf("  Gold (G), Silver (S), or Bronze (B): ");
    scanf("%c", &type);

    printf("Enter the amount($) ");
    scanf("%f", &amount);

    if (type == 'G') {
        if (amount >= 500 && amount < 1000)
            amount = amount * 0.8;
        else if (amount >= 1000)
            amount = amount * 0.7;
    } else if (type == 'S') {
        if (amount >= 1000) {
            amount = amount * 0.8;
        }
    } else if (type == 'B') {
        if (amount >= 2000) {
            amount = amount * 0.9;
        }
    }

    printf("Amount to pay is $%f\n", amount);
    fflush(stdin);
    getchar();
}
```

Each `char` type variable can hold one character.

Remember it.

Constant `char` type values are represented with a pair of single quotes, such as 'A' or 'G'.
There is no reason why we cannot put an `if` structure within another `if` structure.

```java
if (type == 'G') {
    if (amount >= 500 && amount < 1000)
        amount = amount * 0.8;
    else if (amount >= 1000)
        amount = amount * 0.7;
} else if (type == 'S') {
    if (amount >= 1000) {
        amount = amount * 0.8;
    }
} else if (type == 'B') {
    if (amount >= 2000) {
        amount = amount * 0.9;
    }
}
```

The first level `if` structure checks the membership type and the second level `if` structure checks the purchase amount.

Study the above program and understand that it satisfies the required membership discount scheme.
The **multi-level if-else structure** could be rewritten as a **single level if-else structure**.

```java
...
if ((type == 'G') && (amount >= 500 && amount < 1000))
{
    amount = amount * 0.8;
}
else if ((type == 'G') && (amount >= 1000)) {
    amount = amount * 0.7;
}
else if ((type == 'S') && (amount >= 1000)) {
    amount = amount * 0.8;
}
else if ((type == 'B') && (amount >= 2000)) {
    amount = amount * 0.9;
}
...
```

**Learn this trick.**

The trick is to **combine** the **level 1 and level 2 conditions** using the **AND operator &&**. Any amount calculation statement is executed only if **both of the conditions are evaluated to true (non-zero)**. This is exactly what the **AND operator &&** is capable of, evaluating to **one** if **both values are true (non-zero)**.
Achieve Target 3

Many of the round brackets are not needed, but they are put in to make it clearer to read. It is also a good practice to make sure that the desired operators are executed first.

```java
... 
if ((type == 'G') && (amount >= 500 && amount < 1000)) {
    amount = amount * 0.8;
} 
else if ((type == 'G') && (amount >= 1000)) {
    amount = amount * 0.7;
} 
else if ((type == 'S') && (amount >= 1000)) {
    amount = amount * 0.8;
} 
else if ((type == 'B') && (amount >= 2000)) {
    amount = amount * 0.9;
} 
... 
```
The processing part of a Taxi Fare Calculator is shown below.

```c
... char area;
    float distance;
    float fare;
...
    if (area == 'N') {
        if (distance < 2.0)
            fare = 12.5;
        else {
            fare = 12.5 + (distance - 2.0) * 5.0;
        }
    }
    else if (area == 'L') {
        if (distance < 2.5)
            fare = 12.5;
        else {
            fare = 12.5 + (distance - 2.5) * 4.0;
        }
    }
    else if (area == 'H' || area == 'K') {
        if (distance < 2.0)
            fare = 15;
        else {
            fare = 15 + (distance - 2.0) * 7.0;
        }
    }
...
Rewrite the **if-else multi-level structure** to a **single level (flat) if-else structure**. **Work** on it to **make it happen**.

```c
... char area;
    float distance;
    float fare;
...

if (area == 'N') {
    if (distance < 2.0)
        fare = 12.5;
    else {
        fare = 12.5 + (distance - 2.0) * 5.0;
    }
}
else if (area == 'L') {
    if (distance < 2.5)
        fare = 12.5;
    else {
        fare = 12.5 + (distance - 2.5) * 4.0;
    }
}
else if (area == 'H' || area == 'K') {
    if (distance < 2.0)
        fare = 15;
    else {
        fare = 15 + (distance - 2.0) * 7.0;
    }
}
...
```

**Remember** to use round brackets to make sure the desired operators are executed first.
Achieve Target 3

Solution

```c
...
    if ((area == 'N') && (distance < 2.0))
        fare = 12.5;
    else if ((area == 'N') && (distance >= 2.0))
        fare = 12.5 + (distance - 2.0) * 5.0;
    else if ((area == 'L') && (distance < 2.5))
        fare = 12.5;
    else if ((area == 'L') && (distance >= 2.5))
        fare = 12.5 + (distance - 2.5) * 4.0;
    else if ((area == 'H' || area == 'K') && (distance < 2.0))
        fare = 15;
    else if ((area == 'H' || area == 'K') && (distance >= 2.0))
        fare = 15 + (distance - 2.0) * 7.0;
...
```
Achieve Target 3

This time you should find that the rewritten version is cumbersome.

```java
... if ((area == 'N') && (distance < 2.0))
    fare = 12.5;
else if ((area == 'N') && (distance >= 2.0))
    fare = 12.5 + (distance - 2.0) * 5.0;
else if ((area == 'L') && (distance < 2.5))
    fare = 12.5;
else if ((area == 'L') && (distance >= 2.5))
    fare = 12.5 + (distance - 2.5) * 4.0;
else if ((area == 'H' || area == 'K') && (distance < 2.0))
    fare = 15;
else if ((area == 'H' || area == 'K') && (distance >= 2.0))
    fare = 15 + (distance - 2.0) * 7.0;
...```

In most cases, multi-level if-else structure is quicker to execute and easier to read.
Achieve Target 4

We now learn a new structure for selection.

```c
#include <stdio.h>

void main() {
    int number;

    number = 1;
    switch (number) {
        case 0:
            printf("Zero\n");
            break;
        case 1:
            printf("One\n");
            break;
        case 2:
            printf("Two\n");
            break;
        case 3:
            printf("Three\n");
            break;
    }
    getchar();
}
```

The result of the source is:

One
In a selection structure, some statements are executed and others are not.

```c
#include <stdio.h>
void main() {
    int number;
    number = 1;
    switch (number) {
    case 0:
        printf("Zero\n");
        break;
    case 1:
        printf("One\n");
        break;
    case 2:
        printf("Two\n");
        break;
    case 3:
        printf("Three\n");
        break;
    }
    getchar();
}
```

The Output gives a hint that the green statements are executed.

The reasons behind include:

- the switch keyword,
- the variable following it,
- the value following the case keyword,
- and the break keyword.
Achieve Target 4

We now want the program to print “Two”.

```
#include <stdio.h>

void main() {
    int number;
    number = 1;
    switch (number) {
    case 0:
        printf("Zero\n");
        break;
    case 1:
        printf("One\n");
        break;
    case 2:
        printf("Two\n");
        break;
    case 3:
        printf("Three\n");
        break;
    }
    getchar();
}
```

Work on the program, in particular the value of variable number, to make it happen.
Solution

Changing the value of variable `number` from 1 to 2 is a solution.

```java
...  
int number;

number = 2;
switch (number) {
...  
```
A significant feature of the `switch` structure is that it controls the execution of statements by matching cases. The case’s value must be exactly matched by the value following the `switch` keyword, so that the statements following the `case` keyword would be executed.

Remember it.
To illustrate the role of the `break` keyword, we remove them from the program of the previous Target.

```c
#include <stdio.h>
void main() {
    int number;
    number = 1;

    switch (number) {
    case 0:
        printf("Zero\n");

    case 1:
        printf("One\n");

    case 2:
        printf("Two\n");

    case 3:
        printf("Three\n");
    }
    getchar();
}
```

The result of the source is:

```
One
Two
Three
```
Achieve Target 5

The output is different with the `break` keywords removed.

```c
#include <stdio.h>
void main() {
    int number;
    number = 1;

    switch (number) {
    case 0:
        printf("Zero\n");
    case 1:
        printf("One\n");
    case 2:
        printf("Two\n");
    case 3:
        printf("Three\n");
    }
    getchar();
}
```

By observation of the output, the statements executed are in green.

Because the value of variable `number` is 1, so the case that matches is case 1. The printing of “One” is therefore expected.

The unexpected printing of “Two” and “Three” suggests that the `break` keyword has an effect of the execution.
The following program is a modification of the one in the previous Target.

```c
#include <stdio.h>

void main() {

    int number;

    number = 1;
    switch (number) {
    case 3:
        printf("Three\n");

    case 2:
        printf("Two\n");

    case 1:
        printf("One\n");

    case 0:
        printf("Zero\n");
    }
    getchar();
}
```

The result of the source is

```
One
Zero
```
Again the printing of “One” is normal and expected because the value of number matches the case value (which is 1).

```c
#include <stdio.h>
void main() {
    int number;
    number = 1;
    switch (number) {
        case 0:
            printf("Zero\\n");
            break;
        case 1:
            printf("One\\n");
            break;
        case 2:
            printf("Two\\n");
            break;
        case 3:
            printf("Three\\n");
            break;
    }
    getchar();
}
```

The printing of “Zero” is however interesting. Why?

The result of the source is

```
One
Zero
```
Achieve Target 5

Guess the output of the following program.

```c
...  
    number = 1;
    switch (number) {
        case 2:
            printf("Two\n");
        case 1:
            printf("One\n");
        case 0:
            printf("Zero\n");
        case 3:
            printf("Three\n");
    }
...  
```
Achieve Target 5

Solution

One
Zero
Three
Achieve Target 5

Describe the effect of the `break` keyword and how the `switch` structure is executed in general.

```c
...  
    number = 1;
    switch (number) {
        case 2:
            printf("Two\n");
        case 1:
            printf("One\n");
        case 0:
            printf("Zero\n");
        case 3:
            printf("Three\n");
    }
...  
```

One
Zero
Three
The table below summarises the significance of some **typical features** found in a **switch** structure.

<table>
<thead>
<tr>
<th>switch (variable)</th>
<th>The beginning of a switch structure. The value of the variable will be considered in the cases following it in the structure. The execution of the program will break and move to the matched case.</th>
</tr>
</thead>
<tbody>
<tr>
<td>case 1:</td>
<td>The execution will be moved at this point if the value of the switch variable matches the case’s value.</td>
</tr>
<tr>
<td>break;</td>
<td>The execution will be moved out from this point to the end of the switch structure.</td>
</tr>
</tbody>
</table>

**Remember it.**

It is important to note that the **case** keyword should be seen only as a signpost and it has no other effect. **When an execution passes through a case keyword, nothing will happen.** This explains why after printing “One”, the execution would move on to print “Zero” by passing through **case 0**.
The following diagram illustrates the execution of a switch structure.
Another important point to know is that the `switch` variable cannot be a `float` type variable.

<table>
<thead>
<tr>
<th>switch (variable)</th>
<th>The beginning of a switch structure. The value of the variable will be considered in the cases following it in the structure. The execution of the program will break and move to the matched case.</th>
</tr>
</thead>
<tbody>
<tr>
<td>case 1:</td>
<td>The execution will be moved at this point if the value of the switch variable matches the case’s value.</td>
</tr>
<tr>
<td>break;</td>
<td>The execution will be moved out from this point to the end of the switch structure.</td>
</tr>
</tbody>
</table>

You could use `int` type or `char` type variable in `switch`. Remember it.
The following program checks for the entered character and reports if it is a vowel in the English alphabets.

```c
#include <stdio.h>

void main() {
    char letter;

    printf("Enter a lower-case letter: ");
    scanf("%c", &letter);

    switch (letter) {
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u':
            printf("The letter is a vowel\n");
            break;
        case 'y':
            printf("The letter is often treated as a vowel\n");
            break;
        case 'b':
        case 'c':
        case 'd':
        case 'f':
        case 'g':
        case 'h':
        case 'j':
        case 'k':
        case 'l':
        case 'm':
        case 'n':
        case 'p':
        case 'q':
        case 'r':
        case 's':
        case 't':
        case 'v':
        case 'w':
        case 'x':
        case 'z':
            printf("The letter is not a vowel\n");
            break;
        default:
            printf("Entered is not a lower-case letter\n");
            break;
    }

    fflush(stdin);
    getchar();
}
```
Achieve Target 6

The **default** keyword is really a case that is executed if none of the case specified matches. If the character entered does not match any one of the cases specified, then it must not be a lower case letter.

```c
#include <stdio.h>

void main() {
    char letter;
    printf("Enter a lower-case letter: ");
    scanf("%c", &letter);

    switch (letter) {
        case 'a':
        case 'e':
        case 'i':
            case 'o':
            case 'u':
                printf("The letter is a vowel\n");
                break;
        case 'y':
            printf("The letter is often treated as a vowel\n");
            break;
        default:
            printf("Entered is not a lower-case letter\n");
        }
    }
    fflush(stdin);
    getchar();
}
```

Remember it.

The **default** keyword is useful in catching error values.
Achieve Target 6

Note that **C programs** ignore the **line structure** generally. 
A **line break** is treated as a **space**.

```c
#include <stdio.h>
void main() {
    char letter;
    printf("Enter a lower-case letter: ");
    scanf("%c", &letter);

    switch (letter) {
    case 'a':
    case 'e':
    case 'i': case 'o': case 'u':
        printf("The letter is a vowel\n");
        break;
    case 'y':
        printf("The letter is often treated as a vowel\n");
        break;
    case 'b': case 'c': case 'd':
    case 'f': case 'g': case 'h':
    case 'j': case 'k':
    case 'l': case 'm': case 'n':
    case 'p': case 'q': case 'r':
    case 's':
    case 't': case 'v': case 'w':
    case 'x':
    case 'z':
        printf("The letter is not a vowel\n");
    default:
        printf("Entered is not a lower-case letter\n");
    }
    fflush(stdin);
    getchar();
}
```

Remember it.

*It doesn’t matter how you arrange the cases in one per line or several per line.*
Now base on the below program, write a program that reports and integer (from 1 to 10) whether it is a prime number or not a prime number.

```c
#include <stdio.h>

void main() { 
    char letter;
    printf("Enter a lower-case letter: ");
    scanf("%c", &letter);

switch (letter) {
    case 'a':
    case 'e':
    case 'i': case 'o': case 'u':
        printf("The letter is a vowel\n");
        break;
    case 'y':
        printf("The letter is often treated as a vowel\n");
        break;
    case 'b': case 'c': case 'd':
    case 'f': case 'g': case 'h':
    case 'j': case 'k':
    case 'l': case 'm': case 'n':
    case 'p': case 'q': case 'r':
    case 's':
    case 't': case 'v': case 'w':
    case 'x':
    case 'z':
        printf("The letter is not a vowel\n");
    break;
    default:
        printf("Entered is not a lower-case letter\n");
        fflush(stdin);
        getchar();
}
```

The number 1 is a special number. Numbers 2, 3, 5, and 7 are prime numbers. The rest are not prime numbers. Work on it to make it happen.
#include <stdio.h>

void main() {
    int number;

    printf("Enter an integer (1 to 10): ");
    scanf("%d", &number);

    switch (number) {
    case 1:
        printf("The number is special\n");
        break;
    case 2: case 3: case 5: case 7:
        printf("The number is a prime number\n");
        break;
    case 4: case 6: case 8: case 9: case 10:
        printf("The number is not a prime number\n");
        break;
    default:
        printf("Entered is not an integer between 1 to 10\n");
    }

    fflush(stdin);
    getchar();
}
Achieve Target 6

After you have written and understood the program, memorize the program.

```c
#include <stdio.h>

void main() {
    int number;

    printf("Enter an integer (1 to 10): ");
    scanf("%d", &number);

    switch (number) {
    case 1:
        printf("The number is special\n");
        break;
    case 2: case 3: case 5: case 7:
        printf("The number is a prime number\n");
        break;
    case 4: case 6: case 8: case 9: case 10:
        printf("The number is not a prime number\n");
        break;
    default:
        printf("Entered is not an integer between 1 to 10\n");
    }

    fflush(stdin);
    getchar();
}
```
On a white sheet of paper, write the Prime Number Checker program (from 1 to 10) again without looking at any reference material.
In the last 6 targets, you have touched on the following ideas.

In the last 6 targets, you have worked hard on the following ideas. Ponder upon these ideas and remember what you have learned.
Achieve Target 7

1. The **if** keyword can control whether to execute a statement. You can add an **else** part to an **if** structure. In an **if-else** structure, if the value following the **if** keyword is **non-zero**, the statement following **if** is executed. If the value is **zero**, the statement following **else** is executed.

2. **Comparing to if structures, using if-else structures generally makes program executes less operators and therefore faster.**

3. You can put an **if** structure within another **if** structure. This multi-level **if** structure is called **nested if** structure or simply **nesting**. Proper nesting again generally makes program executes less operators and therefore faster.

4. The **char** type variable/values can hold one ASCII character. ASCII is a common standard used in representing common characters, digits, and symbols in digital form. So one such character, digit, etc is represented as a number from 0 to 255. A **char** type variable can hold a value from 0 to 255.
5. A char type constant value should be enclosed within a pair of single quotes. So the character \texttt{a} is represented in a C program as \texttt{\'a\}'. The conversion character for char variable/values in \texttt{printf} and \texttt{scanf} is \texttt{%c}. Remember to use the matched conversion character (\texttt{%d} for int and \texttt{%f} for float).

6. The \texttt{switch} selection structure provides an alternative to the \texttt{if-else} structure for conditional execution of statements. The \texttt{switch} structure can only test the cases of one int or char type variable, and is therefore not as flexible as the \texttt{if-else} structure. All \texttt{switch} structures could be re-written as \texttt{if-else} structure, but not the reverse. The \texttt{switch} structure however makes programs easier to read.

7. The \texttt{break} keyword plays a pivotal role in regulating the execution in a \texttt{switch} structure. It specifies the exit points where execution leaves the \texttt{switch} structure, and otherwise the execution would continue. You will also find the \texttt{break} keyword in other structures that we will study later. The effect is however the same, causing the execution to leave the structure at the point.
End of Session 3

You have done well! Keep going!

Session 4 26 November 2002