Logic and Selection 1

Session 2  
29 October 2002
The following program evaluates how a sum of money can be converted to a number of $500 bank notes and the remaining amount.

```c
#include<stdio.h>
void main() {
int amount;
int notesCount;
int remainder;

    printf("Enter an amount of money ");
    scanf("%d", &amount);

    notesCount = amount / 500;
    remainder = amount % 500;

    printf("Converted to %d $500 notes and remainder is %d\n", 
            notesCount, remainder);
    fflush(stdin);
    getchar();
}
```

The program uses type `int` variables and a new operator '%'. The '%' operator can be applied to `int` type variables to evaluate the remainder of a quotient.
We now want another program to work out the number of tables for a given number of participants in a wedding banquet. Each table has a size of 12. The program should print the minimal number of tables needed and the remaining number of people.

```c
#include<stdio.h>

void main() {
    int amount;
    int notesCount;
    int remainder;

    printf("Enter an amount of money ");
    scanf("%d", &amount);
    
    notesCount = amount / 500;
    remainder = amount % 500;

    printf("Converted to %d $500 notes and remainder is %d\n", 
            notesCount, remainder);
    fflush(stdin);
    getchar();
}
```

Work on the program to make it happen.
Achieve Target 1

Solution

```c
#include<stdio.h>

void main() {
    int people;
    int tableCount;
    int remainder;

    printf("Enter number of people ");
    scanf("%d", &people);

    tableCount = people / 12;
    remainder = people % 12;

    printf("Minimum %d tables and remainder is %d\n", tableCount, remainder);
    fflush(stdin);
    getchar();
}
```
Achieve Target 1

The calculation results or values can be directly given to the printf statement.

**Discuss** the advantages and disadvantages by studying the following program.

```c
void main() {
    int amount;

    printf("Enter an amount of money ");
    scanf("%d", &amount);

    printf("Converted to %d $500 notes and remainder is %d\n", amount / 500, amount % 500);

    fflush(stdin);
    getchar();
}
```
The following program allows us to see the effects of some common operators. The application of each operator will produce a value.

```c
#include<stdio.h>
void main() {
    int a = 25;
    int b = 10;
    printf("a + b is %d\n", a + b);
    printf("a - b is %d\n", a - b);
    printf("a * b is %d\n", a * b);
    printf("a / b is %d\n", a / b);
    printf("a % b is %d\n", a % b);
    getchar();
}
```

```
a + b is 35
a - b is 15
a * b is 250
a / b is 2
a % b is 5
```
Note that \%d is used to print int type values/variables and \%f is used to print float type values/variables. The characters \%\% is used to print the character %.

```c
#include<stdio.h>
void main() {
    int a = 25;
    int b = 10;
    printf("a + b is %d \n", a + b);
    printf("a - b is %d \n", a - b);
    printf("a * b is %d \n", a * b);
    printf("a / b is %d \n", a / b);
    printf("a % b is %d \n", a % b);
    getchar();
}
```

Remember it.
Achieve Target 2

**Work** and make guesses to the output of the program.

```c
#include<stdio.h>
void main() {
    int a = 25;
    int b = 10;

    printf("a < b is %d\n", a < b);
    printf("a > b is %d\n", a > b);
    printf("a <= b is %d\n", a <= b);
    printf("a >= b is %d\n", a >= b);
    printf("a == b is %d\n", a == b);

getchar();
}
```

**Remember:** The application of each operator will produce a value.
Achieve Target 2

Solution

\[
\begin{align*}
a < b & \text{ is 0} \\
a > b & \text{ is 1} \\
a \leq b & \text{ is 0} \\
a \geq b & \text{ is 1} \\
a == b & \text{ is 0}
\end{align*}
\]

How many answers you get right?
What is the meaning of \( 0 \) and \( 1 \)?

\[
\begin{align*}
a &= 25 \\
b &= 10
\end{align*}
\]

- \( a < b \) is 0
- \( a > b \) is 1
- \( a \leq b \) is 0
- \( a \geq b \) is 1
- \( a == b \) is 0

What is the difference between = and ==?
Achieve Target 2A

Work and make guesses to the output of the program.

```c
#include<stdio.h>

void main() {
    int a = 25;
    int b = 10;

    printf("a == b is %d
", a == b)
    printf("a != b is %d
", a != b);
    printf("a and b is %d
", a && b);
    printf("a or is %d
", a || b);
    printf("a = b = 30 is %d
", a = b = 30);
    printf("value of a and b are %d %d
", a, b);

    getchar();
}
```

Remember: The application of each operator will produce a value.
Achieve Target 2A

Solution

\[
\begin{align*}
a &= b & \text{is 0} \\
a \neq b & \text{is 0} \\
a && b & \text{is 1} \\
a || b & \text{is 1} \\
a = b = 30 & \text{is 30} \\
\text{value of a and b are 30 30}
\end{align*}
\]

Do you get the right answers?
Achieve Target 2A

When will \( a \text{ and } b \ (a \&\& b) \) give the value 0?

How about \( a \text{ or } b \ (a \|\| b) \) ?

\[
\begin{align*}
a \text{ == } b & \text{ is 0} \\
a \text{ !} \neq b & \text{ is 1} \\
a \&\& b & \text{ is 1} \\
a \|\| b & \text{ is 1} \\
a = b = 30 & \text{ is 30}
\end{align*}
\]

value of a and b are 30 30

What’s happen to \( a \) and \( b \) when we write \( a = b = 30 \) ?
A brand of soft-drink has launched a promotion campaign. One free can is given for every five cans purchased. Also if a customer purchased 30 or more cans, an additional 3 cans are given freely. The following program is used in a supermarket to calculate the number of free cans that should be given to a customer.

```c
#include <stdio.h>
void main() {

    int numberOfCan;
    int freeCan;

    printf("Enter the number of cans purchased: ");
    scanf("%d", &numberOfCan);
    freeCan = numberOfCan / 5 + (numberOfCan >= 30) * 3;

    printf("The customer should receive extra %d cans free\n", freeCan);
    fflush(stdin);
    getchar();
    getchar();
}
```
Achieve Target 3

One free can is given for every five cans purchased. Also if a customer purchased 30 or more cans, an additional 3 cans are given freely.

```c
#include <stdio.h>
void main() {

    int numberOfCan;
    int freeCan;

    printf("Enter the number of cans purchased: ");
    scanf("%d", &numberOfCan);
    freeCan = numberOfCan / 5 + (numberOfCan >= 30) * 3;

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

    fflush(stdin);
    getchar();
}
```

Note that the round brackets are used where the expressions should be evaluated first.

Learn that it is a better practice to add them even if not needed.

The lines in red do the calculation using several operators. Discuss how it happens.
Use the wedding banquet program you have written for Target 1, **work** on it so that it calculates the actual number of tables required by checking the number of remaining people. If the remaining people number is more than 3, then one more table should be added to the total number. Otherwise, the number of table stays the same (by squeezing the extra people in).

```c
#include <stdio.h>
void main() {

    int numberOfCan;
    int freeCan;

    printf("Enter the number of cans purchased: ");
    scanf("%d", &numberOfCan);
    freeCan = numberOfCan / 5 + (numberOfCan >= 30) * 3;

    printf("The customer should receive extra %d cans free\n", freeCan);
    fflush(stdin);
    getchar();
}
```
Solution

```c
#include<stdio.h>

void main() {

    int people;
    int tableCount;

    printf("Enter number of people ");
    scanf("%d", &people);

    tableCount = people / 12 + (people % 12 > 3);

    printf("Actual number of tables is %d\n", tableCount);
    fflush(stdin);
    getchar();
}
```
Achieve Target 4

We now leave the operators for a moment and study something new.

```c
#include<stdio.h>

void main() {
    if (0)
        printf("One\n");

    if (1)
        printf("Two\n");

    if (2)
        printf("Three\n");

    if (-1)
        printf("Four\n");

    if (0) {
        printf("Five\n");
        printf("Six\n");
    }

    printf("Seven\n");
    getchar();
}
```

The result of the source is:

```
Two
Three
Four
Seven
```
The statements in red are clearly not executed.

The reason is the `if` keyword and the value followed in the round bracket.

If the value following the `if` keyword is zero, then the following statement is not executed. If the value is non-zero, the following statement is executed.

Work and make a guess on which values will allow the execution to go through the statement, and which values will prevent the execution to happen.
We now want the program to write "One", "Five" and "Six". Work on the program to make it happen.

```c
#include<stdio.h>
void main() {
    if (0)
        printf("One\n");
    if (1)
        printf("Two\n");
    if (2)
        printf("Three\n");
    if (-1)
        printf("Four\n");
    if (0) {
        printf("Five\n");
        printf("Six\n");
    }
    printf("Seven\n");
    getchar();
}
```
Solution

To rewrite the program to write "One", "Five", and "Six", the method is to change the values following the if keywords. Change it to zero if the printing is not desired. Change it to non-zero if printing is desired. To not print the "Seven", an addition if keyword could be added or simply remove the statement.

```c
#include<stdio.h>
void main() {
    if (1)
        printf("One\n");
    if (0)
        printf("Two\n");
    if (0)
        printf("Three\n");
    if (0)
        printf("Four\n");
    if (1) {
        printf("Five\n");
        printf("Six\n");
    }
    if (0)
        printf("Seven\n");
    getchar();
}
```
```c
#include<stdio.h>

void main() {
    if (1)
        printf("One\n");
    if (0)
        printf("Two\n");
    if (0)
        printf("Three\n");
    if (0)
        printf("Four\n");

    if (1) {
        printf("Five\n");
        printf("Six\n");
    }

    if (0)
        printf("Seven\n");
    getchar();
    getchar();
}
```

Do you notice that the two statements printing "Five" and "Six" seem to be treated as one statement in the execution?

Remember it.

Putting any number of statements between curly brackets makes the statements into one statement.
Achieve Target 5

We now know that the value in the if statement can control the execution. We could rewrite the free can calculator in Target 3 to use if statement to see whether 3 more cans should be given freely if 30 or more cans are purchased. Compare the following program with that in Target 3.

```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);
    fflush(stdin);
    getchar();
}
```
Achieve Target 5

One free can is given for every five cans purchased. Also if a customer purchased 30 or more cans, an additional 3 cans are given freely.

```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);
    fflush(stdin);
    getchar();
}
```

The **curly brackets** are not necessary because only one statement is to be controlled, but having the brackets is a better practice.

The **if** statement controls the execution of one statement that adds three more free cans.

Remember it.
Achieve Target 5

Take the wedding banquet program you have written in Target 3. Modify the program so that it uses the if statement to evaluate whether an extra table is added. Work on the program to make it happen.

```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);
    fflush(stdin);
    getchar();
}
```

Each table has a size of 12.
If the remaining people number is more than 3, then one more table should be added to the total number. Otherwise, the number of table stays the same (by squeezing the extra people in).
Solution

```c
#include<stdio.h>

void main() {

    int people;
    int tableCount;

    printf("Enter number of people ");
    scanf("%d", &people);

    tableCount = people / 12;

    if (people % 12 > 3) {
        tableCount++;
    }

    printf("Actual number of tables is %d\n", tableCount);
    fflush(stdin);
    getchar();
}
```
Achieve Target 6

This is the final program of the previous Project Perform Session. This program calculates BMI from the input height and weight.

```c
#include<stdio.h>

void main(){

    float height, weight, bmi;

    printf("Enter your Height(m): ");
    scanf("%f", &height);

    printf("Enter your Weight(kg): ");
    scanf("%f", &weight);

    bmi = weight / (height * height);

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

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

In addition to printing the BMI value, modify the program so that it prints the health status of the person according to the following table. Work on the program to make it happens. You will need four `if` statements. You need one for each type of assessment.

```c
#include<stdio.h>
void main(){
    float height, weight, bmi;

    printf("Enter your Height(m): ");
    scanf("%f", &height);

    printf("Enter your Weight(kg): ");
    scanf("%f", &weight);

    bmi = weight / (height * height);

    printf("Your BMI = %f\n", bmi);
    fflush(stdin);
    getchar();
}
```

<table>
<thead>
<tr>
<th>BMI</th>
<th>Assessment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less than 19</td>
<td>Underweight</td>
</tr>
<tr>
<td>Between 19 to 22 (inclusive)</td>
<td>Just Right</td>
</tr>
<tr>
<td>Between 22 (exclusive) and 25 (inclusive)</td>
<td>Overweight</td>
</tr>
<tr>
<td>Over 25</td>
<td>Health at Risk</td>
</tr>
</tbody>
</table>
#include<stdio.h>

/* Rewrite the BMI calculator with health evaluation */

void main(){
    float height, weight, bmi;

    printf("BMI calculator\n");
    printf("Enter your Height(m): ");
    scanf("%f", &height);

    printf("Enter your Weight(kg): ");
    scanf("%f", &weight);

    bmi = weight / (height * height);
    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");

    fflush(stdin);
    getchar();
}

Achieve Target 6

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

```c
#include<stdio.h>

/* Rewrite the BMI calculator with health evaluation */

void main(){
    float height, weight, bmi;
    printf("BMI calculator\n");

    printf("Enter your Height(m): ");
    scanf("%f", &height);

    printf("Enter your Weight(kg): ");
    scanf("%f", &weight);

    bmi = weight / (height * height);
    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");

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

On a white sheet of paper, **write** the Health Evaluation program again without looking at any reference material.

<table>
<thead>
<tr>
<th>BMI</th>
<th>Assessment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less then 19</td>
<td>Underweight</td>
</tr>
<tr>
<td>Between 19 to 22 (inclusive)</td>
<td>Just Right</td>
</tr>
<tr>
<td>Between 22 (exclusive) and 25 (inclusive)</td>
<td>Overweight</td>
</tr>
<tr>
<td>Over 25</td>
<td>Health at Risk</td>
</tr>
</tbody>
</table>
Achieve Target 7

In the last 6 targets, you have touched on the following ideas.

Ponder upon these ideas and remember what you have learned.

The ideas are arranged so that the next one is more challenging. We expect you at least reach to Idea 6.
1. The four common elements of C programs – input, output, variables (memory) and operators. These four appears in the program (from Target 4) that you have memorized. Could you identify them?

2. The '%' operator can be applied to int values or variables to find the remainder. It cannot be applied to float values or variables.

3. All operators and variables will evaluate to a value. In addition to this effect, the following three operators '='; '++', and '--' also changes the value in a variable. The operator '=' assigns the value on the right to the variable on the left. The operator '++' increases the variable on either side by one. The operator '--' decreases the variable on either side by one.

4. The use of round brackets ( ) can control the evaluation order of the operators. The operators have a pre-set order of evaluation depending on their precedence. The round brackets forces the evaluation of the operations enclosed within. It is a good practice to put brackets to force the evaluation if you are not sure of the precedence.
5. The `if` keyword can control whether to execute a statement. If the value following the `if` keyword is zero, the statement is not executed. If the value is non-zero, the statement is executed.

6. The use of curly brackets `{   }` can make multiple statements into one statement, which is called a **compound statement**. This allows one `if` keyword to control the execution of multiple statements.

7. The `if` keyword allows what is known as a **selection structure**. If logic allows C programs to process data and turn it into information. The selection structure allows the information to be more intelligent. The BMI program can perform an evaluation of health with the selection structure. When you read a C program, identify the parts that makes the C program intelligent.
End of Session 2

You have done well! Keep going!

Session 3  12 November 2002