module2

gar

Outline

Branching

- When an algorithm makes a choice to do one of two or more things, it's called branching
- Different options are available to make choices
  - `if` : to conditionally execute the statements in its block (the statements written between `{ ... }` after the `if` keyword)
  - `if-else` : to make two-way decisions
  - Cascaded `if-else` : for multi-way decision
  - Nested `if-else` : branching within branching
  - `Switch` : making multi-way decisions in another way

Two-way Selection: If

- Required to make decisions
- E.g. To decide whether a person is senior citizen, we check the age

```
if (age >= 60)
  printf("Person is a senior citizen");
```

- In the example, we are printing the statement only when the expression is true

Two-way Selection: If

- Whenever we require more than one statement to be executed within the `if` block, we need the braces. Otherwise, we may leave out the braces

```
if (age >= 60)
{
  printf("Person is a senior citizen");
  ticket_cost = 0.9*ticket_cost; /* Give a discount */
}
```

- If the braces were removed, everybody gets a discount and the ticket seller's profit would be reduced

Two-way Selection: If-Else

- The if-else statement tells what to do when an expression is true and what to do when it's false
- The syntax is

```
if (expression)
  statement_1
else
  statement_2
```

- E.g. To print whether a number is even or odd

```
if (num%2 == 0)
  printf("number is even");
else
  printf("number is odd");
```

Multi-way decision: Else-If (Cascaded if-else)

- The construction of a multiway decision is written as

```
if (expression1)
  statement1
else if (expression2)
  statement2
else if (expression3)
  statement3
else
  statementn
```

- As soon as one of the expressions holds true, the statements inside the body is executed and goes out of the `else-if` chain
- The `else` part serves as a default, when none of the given expressions is true

Multi-way decision: Else-If

- E.g. To display the grade obtained by the student in an exam

```
if (marks >= 90)
  printf("A");
else if (marks >= 80 && marks < 90)
  printf("B");
else if (marks >= 70 && marks < 80)
  printf("C");
else if (marks >= 60 && marks < 70)
  printf("D");
else if (marks >= 50 && marks < 60)
  printf("E");
else
  printf("F");
```

Two-way Selection: Nested If-Else

- E.g. Displaying maximum of three numbers

```
if (a > b) {
  if (a > c)
    max = a;
  else
    max = c;
}
else {
  if (b > c)
    max = b;
  else
    max = c;
}
```

Switch Statement

- This is another multi-way decision that tests whether an expression matches one of a number of constant integer values, and branches accordingly

```
switch (expression) {
  case constant1: statements
                  break;
  case constant2: statements
                  break;
  default: statements
}
```

- If the expression is `constant1`, the statements in front of that number is executed, and the `break` gets the execution flow out of the switch block

Switch Statement – Example 1

- A simple calculator: enter two numbers and an operator to perform the required arithmetic operation

```
int a,b;
char op;
scanf("%d%d", &a, &b);
scanf("%c", &op);
switch (op)
{
  case '+': printf("%d\n", a+b);
            break;
  case '-': printf("%d\n", a-b);
            break;
  case '*': printf("%d\n", a*b);
            break;
  case '/': printf("%f\n", (float)a/b);
            break;
  default: printf("Enter an arithmetic operator\n");
}
```

Switch Statement – Example 2

- Tell whether the entered character is a vowel or not

```
char ch;
scanf("%c", &ch);
switch (ch)
{
  case 'a':
  case 'e':
  case 'i':
  case 'o':
  case 'u': printf("%c is a vowel\n"); break;
  default : printf("%c is not a vowel\n");
}
```

- If upper case letters are required, that may be added with the corresponding case labels
- If a statement is missing after a case constant, it will carry on the execution from the first statement it sees. In this example, if 'a' is entered, it executes the first printf statement

Switch Statement – Example 3

- Rewriting the grading example using switch instead of else-if

```
switch (marks / 10)
{
  case 9: printf("A\n"); break;
  case 8: printf("B\n"); break;
  case 7: printf("C\n"); break;
  case 6: printf("D\n"); break;
  case 5: printf("E\n"); break;
  default: printf("F\n");
}
```

Ternary Operator ?:

- ? : is called a ternary operator since it takes three expressions
- Syntax is $expr_1$? $expr_2$: $expr_3$
- If the expression $expr_1$ is true (non-zero), then $expr_2$ is evaluated. Otherwise, $expr_3$ is evaluated.

Ternary Operator ?: – Example

- To compute the max of two numbers, these two code samples are equivalent
- Using if-else

```
if (a>b)
  max = a;
else
  max = b;
```

- Using ternary operator

```
max = (a>b) ? a : b;
```

Loops

- Loops are required when we want to do certain repetetive tasks
- E.g. Printing the squares of first five numbers without loop would require us to write 5 statements

```
main()
{
  printf("%d\n", 1*1);
  printf("%d\n", 2*2);
  printf("%d\n", 3*3);
  printf("%d\n", 4*4);
  printf("%d\n", 5*5);
}
```

- There are different kinds of loops available in C to print that in fewer lines (which is shown after the description of the syntax)

Loops: while

```
while (expression) {
  statements
}
```

- Using if and goto:

```
loop1:
if (expression) {
  statements
  goto loop1;
}
```

Loops: do–while

- The do-while loop is written as:

```
do {
statements
} while (expression);
```

- Note the semicolon after while. Missing that will cause a syntax error
- Equivalently, using if and goto:

```
loop1:
  statements
if (expression)
  goto loop1;
```

Loops: for

- for loop is written as:

```
for (expr1 ; expr2 ; expr3) {
  statements
}
```

- Can be written as an equivalent while loop

```
expr1;
while (expr2) {
  statements
  expr3
}
```

Loops (example) – Squares of numbers

- Continuing from the example, printing the squares can be done as follows (do-while)

```
int i = 1;
do {
  printf("%d\n", i*i);
  i++;
} while(i<6);
```

- (while)

```
int i = 1;
while(i<6) {
  printf("%d\n", i*i);
  i++;
}
```

- (for)

```
int i;
for (i=1; i<6; i++)
  printf("%d\n", i*i);
```

break and continue

- `break` gets the control out of the current loop or switch block
- `continue` gets the control directly to the testing of the condition, and begins the next iteration if condition is satisfied
- E.g. Compute the sum of numbers only if positive numbers are entered

```
int a, sum = 0;
while (1) {
  scanf("%d", &a);
  if (a<0)
    break;
  sum += a;
}
```

- Hence, if `1 7 8 3 -4` are entered, it calculates the sum of first 4 numbers and exits the loop

break and continue

- E.g. Print the first five odd numbers

```
int a=0;
while (a<10) {
  a++;
  if (a%2 == 0)
    continue;
  printf("%d\n", a);
}
```

- In the loop body, whenever a becomes even, `continue` statement is executed, which takes the flow of execution to check the condition a<10 and then continues execution depending on the condition result

goto and labels

- When there is a goto and a label, the statement next to the label gets executed.
- Usually, it's not preferred since it's difficult to read and maintain such code
- Used mainly to exit out of deeply nested loops
- E.g.

```
for ( ... )
  for ( ... )
    for ( ... )
      if (solution_found)
        goto found;
found:
  /* print the solution */
```

- `break` can terminate only one loop, `goto` helps to terminate all the outer loops as well