Selective Structure

Chapter 03

CMPE-112 *Programming Fundamentals*

1

1

Lecture Plan

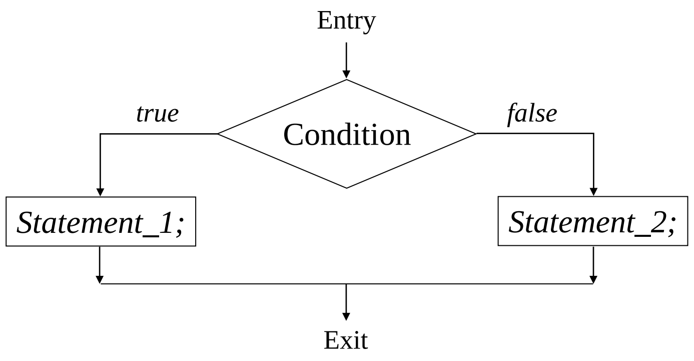
- General Idea
- Relational Operators
- Logical Operators
  - Logical AND operator
  - Logical OR operator
  - Logical NOT operator
  - Precedence and Associativity
- Conditional Expression Operator
- Conditional Statements
  - Statement *if*
  - Statement *if - else*
- Statement *switch*

2

2

General Idea

- *Selective Structure* includes a test for a *condition* followed by *alternative paths* that the program can follow. The program runs along one or the other path depending upon the result of the test for the condition



Relational Operators (I)

- C provide 6 relational operators for comparing values of two expressions

Relational Operator	Name	Relational Expression
<	Less than	*exp1 < exp2*
<=	Less than or equal to	*exp1 <= exp2*
>	Greater than	*exp1 > exp2*
>=	Greater than or equal to	*exp1 >= exp2*
==	Equal to	*exp1 == exp2*
!=	Not equal to	*exp1 != exp2*

Relational Operators (II)

- Relational operators can be applied to operands - of any arithmetic type
- The result of comparison of two expressions is *true* if the condition is satisfied and *false* otherwise
- There is no special logical data type in C. The value of a relational expression is of type *int* :
  - 15 > 10 has the value *1* (*true*)
  - 15 < 10 has the value *0* (*false*)
- Assignment operator "=" vs. equal to operator "=="
  - if (x == 10) printf("equal to operator");
  - if (x = 10) printf("assignment");
- In the latter statement, printing will **always** be performed

Precedence and Associativity

- The precedence and associativity of the relational operators with respect to arithmetic and assignment operators:

Operators	Type	Associativity
+ − ++ −−	Unary	Right to left
* / %	Binary	Left to right
+ −	Binary	Left to right
< <= > >=	Binary	Left to right
== !=	Binary	Left to right
= *= /= %= += −=	Binary	Left to right

Examples (I)

int i = 3, j = 2, k = 1;		
Expression	**Equivalent Expression**	**Value**
i > j > k		
i >= j >= k		
i != j != k		
k < i != k < j		
i - k == j * k		
i > j == i + k > j + k		
i += j != k		
i = k != j < k * j		

7

Correct Answers (I)

int i = 3, j = 2, k = 1;		
Expression	**Equivalent Expression**	**Value**
i > j > k	(i > j) > k	*false*
i >= j >= k	(i >= j) >= k	*true*
i != j != k	(i != j) != k	*false*
k < i != k < j	(k < i) != (k < j)	*false*
i - k == j * k	(i - k) == (j * k)	*true*
i > j == i + k > j + k	(i > j) == ((i + k) > (j + k))	*true*
i += j != k	i += (j != k)	*4*
i = k != j < k * j	i = (k != (j < (k * j)))	*1*

8

Logical Operators

- In C there are three logical operators:
  - Logical AND (&&) - binary
  - Logical OR (||) - binary
  - Logical NOT (!) - unary
- The operands may be of any arithmetic type while the result is always *int*
- The value of a logical expression is either *1* (*true*) or *0* (*false*)

9

9

Logical AND

- The general form is

  exp1 && *exp2*
- Such expression is evaluated from by first evaluating the left expression *exp1*. If its value is 0, the value of *exp2* is not evaluated at all, and the result is *false*

exp1	**exp2**	**exp1 && exp2**
true	true	true
true	false	false
false	true	false
false	false	false

10

10

Logical OR

- The general form is

  exp1 || exp2
- Such expression is evaluated from by first evaluating the left expression *exp1*. If its value is 1, the value of *exp2* is not evaluated at all, and the result is *true*

exp1	**exp2**	**exp1		exp2**
true	true	true
true	false	true
false	true	true
false	false	false

Logical NOT

- The general form is

  ! exp
- Such expression is evaluated from by first evaluating the left expression. If its value is 1, the result is *false*, otherwise the result is *true*

exp	**! exp**
true	false
false	true

Precedence and Associativity

- The precedence and associativity of the logical operators with respect to the others:

Operators	Type	Associativity
+ − ++ −− !	Unary	Right to left
* / %	Binary	Left to right
+ −	Binary	Left to right
< <= > >=	Binary	Left to right
== !=	Binary	Left to right
&&	Binary	Left to right
||	Binary	Left to right
= *= /= %= += −=	Binary	Left to right

Examples (II)

int i = 3, j = 2, k = 1;		
Expression	**Equivalent Expression**	**Value**
!! k		
!i == !j		
k != ! k * k		
i > j && j > k		
i != j && j != k		
i - j - k || k == i / j		
i < j || k < i && j < k		

Correct Answers (II)

int i = 3, j = 2, k = 1;		
Expression	**Equivalent Expression**	**Value**
! ! k	! (! k)	*true*
! i == ! j	(! i) == (! j)	*true*
k != ! k * k	k != ((! k) * k)	*true*
i > j && j > k	(i > j) && (j > k)	*true*
i != j && j != k	(i != j) && (j != k)	*true*
i - j - k || k == i / j	((i - j) - k) || (k == (i / j))	*true*
i < j || k < i && j < k	(i < j) || ((k < i) && (j < k))	*false*

Conditional Expression Operator

- This operator has three arguments

  expression_1 **?** *expression_2* **:** *expression_3*
- The conditional expression is evaluated by first evaluating the *expression_1*. If the resultant value is nonzero (true), then the *expression_2* is evaluated and its value become the overall result. Otherwise, the *expression_3* is evaluated, and its value becomes the result
- This operator is most often used in assignment statements. For example,

  max = x > y **?** *x* **:** *y;*

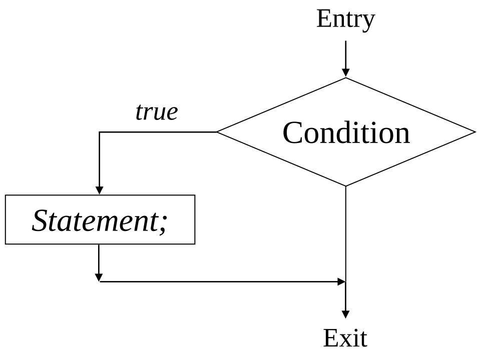
  finds the maximum of two values

Statement *if*

- The general form

  if (expression)

  statement



- Example

```
if ( number < 0 )
  number = -number;
printf("Positive value is %d", number);
```

17

Sample Program (I)

```
#include <stdio.h>

int main()
{
  int v1, v2, max;

  printf("\nEnter two values: ");     /* Enter two numbers */
  scanf("%i %d", &v1, &v2);

  max = v1;     /* Assign the first value as maximum */
  if (v2 > v1)   max = v2;   /* Check is the second number is greater */

  printf("\nMaximum is: %3d\n", max);   /* Print the result */

  return 0;
}
```

18

Statement *if-else*

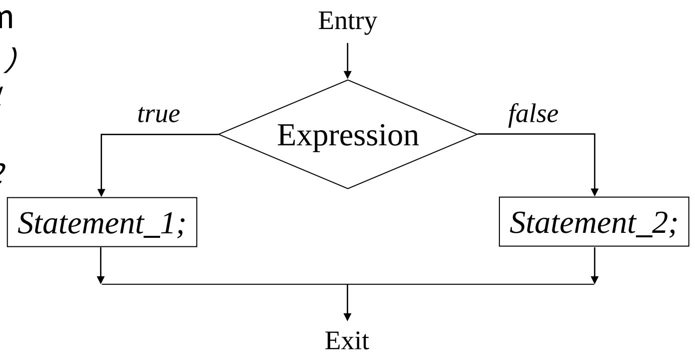
- The general form

  if (expression)

  statement_1

  else

  statement_2



- Example

```
if ( number % 2  ==  0)
  printf("Value of %d is an even number" , number);
else
  printf("Value of %d is an odd number", number);
```

Sample Program (II)

```
#include <stdio.h>

int main()
{
  int v1, v2, max;

  printf("\nEnter two values: ");     /* Enter two numbers */
  scanf("%i %d", &v1, &v2);

  if (v2 > v1)   max = v2;   /* Check is the second number is greater */
  else           max = v1;

  printf("\nMaximum is: %3d\n", max);   /* Print the result */

  return 0;
}
```

Nested Conditional Statements

- Within if-block and/or else-block there may be another if-else statement. Then the general form of a nested conditional statement is as follows:

```
if ( expression_1 )
  if (condition_1)
     statement_1
  else
     statement_2
else
  if (condition_2)
     statement_3
  else
     statement_4
```

- Neither *statement_1* nor *statement_2* is executed unless *expression_1* is **true**
- Neither *statement_3* nor *statement_4* is executed unless *expression_1* is **false**

Sample Program (III)

```
#include <stdio.h>

int main()
{
  int v1, v2;

  printf("\nEnter two values: ");     /* Enter two numbers */
  scanf("%i %d", &v1, &v2);

  /* Print the result */
  if (v1 > v2)    printf("\n%1d is greater than %1d\n", v1, v2);
  else
    if (v1 < v2)  printf("\n%1d is less than %1d\n", v1, v2);
    else          printf("\n%1d is equal to %1d\n", v1, v2);

  return 0;
}
```

Dangling *else* Problem

- When conditional statements are nested, *else* is associated with the **closest** previous *else*-less *if* :

```
if ( expression_1 )
  if (condition_1)
    statement_1
else
  statement_2
```

⇓ *Equivalent*

```
if ( expression_1 )
  if (condition_1)
    statement_1
  else
    statement_2
```

```
if ( expression_1 )
{
  if (condition_1)
    statement_1
}
else
  statement_2
```

There is no ***;*** here

Equivalent Transformation

- Often several expressions with if-else statements can be transformed into one statement using logical operators

```
if ( v1 > v2 )
  if ( v2 > 10 )
    printf("%d is greater than 10", v1);
```

can be transformed like that

```
if ( ( v1 > v2 ) && ( v2 > 10 ) )
  printf("%d is greater than 10", v1);
```

```
i = 0;
if ( v1 > 10 ) i++;
if ( v2 > 10 ) i++;
if ( i > 0 )
  printf("%d or %d > 10", v1, v2);
```

can be transformed like that

```
if ( ( v1 > 10 ) || ( v2 > 10 ) )
  printf("%d or %d > 10", v1, v2);
```

Statement *switch*

- In case of constant multi-way decision C provides a special *switch* statement. Its general form is as follows:

```
switch ( expression )
{
  case value_1 :
        statement_1
        break;
          :
  case value_n :
        statement_n
        break;
  default :
        statement_x
        break;
}
```

$\xrightarrow{\textit{Equivalent}}$

```
if ( expression == value_1 )
   statement_1
else if ( expression == value_2 )
        statement_2
        :
else if ( expression == value_n )
        statement_n
else
      statement_x
```

Sample Program (IV)

```
#include <stdio.h>

int main()
{
  char c1;
  int    v1, v2;

  printf("\nEnter the expression: ");     /* Enter the expression */
  scanf("%i %c %d", &v1, &c1, &v2);

  switch ( c1 ) {  /* Perform the operation requested */
    case '+' : printf("\n%1d plus %1d is %1d\n", v1, v2, v1+v2); break;
    case '–' : printf("\n%1d minus %1d is %1d\n", v1, v2, v1–v2); break;
    default :   printf("\nWrong operation!\n"); break;
  }

  return 0;
}
```