IMPERATIVE PROGRAMMING

UNIT 2
IF CONDITIONAL STATEMENT

Compiled by Ms. Prajakta Joshi
IF....ELSE STATEMENT

Compiled by Ms. Prajakta Joshi
#### ARITHMETIC OPERATORS

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Adds two operands.</td>
<td>A + B = 30</td>
</tr>
<tr>
<td>−</td>
<td>Subtracts second operand from the first.</td>
<td>A − B = -10</td>
</tr>
<tr>
<td>*</td>
<td>Multiplies both operands.</td>
<td>A * B = 200</td>
</tr>
<tr>
<td>/</td>
<td>Divides numerator by denominator.</td>
<td>B / A = 2</td>
</tr>
<tr>
<td>%</td>
<td>Modulus Operator and remainder of after an integer division.</td>
<td>B % A = 0</td>
</tr>
<tr>
<td>++</td>
<td>Increment operator increases the integer value by one.</td>
<td>A++ = 11</td>
</tr>
<tr>
<td>--</td>
<td>Decrement operator decreases the integer value by one.</td>
<td>A-- = 9</td>
</tr>
</tbody>
</table>

Compiled by Ms. Prajakta Joshi
#include <stdio.h>
main()
{
    int a = 21; int b = 10; int c ;
    c = a + b;
    printf("Line 1 - Value of c is %d\n", c );
    c = a - b;
    printf("Line 2 - Value of c is %d\n", c );
    c = a * b;
    printf("Line 3 - Value of c is %d\n", c );
    c = a / b;
    printf("Line 4 - Value of c is %d\n", c );
    c = a % b;
    printf("Line 5 - Value of c is %d\n", c );
}
UNARY OPERATORS IN C

• **Unary operator**: are operators that act upon a single operand to produce a new value.

• **Types of unary operators:**
  - unary minus (-)
  - increment (++)
  - decrement (- -)
  - NOT (!)
  - Addressof operator (&)
  - sizeof()
```c
#include <stdio.h>

int main()
{
    int a = 10, b = 100;
    float c = 10.5, d = 100.5;
    printf("++a = %d \n", ++a);
    printf("--b = %d \n", --b);
    printf("++c = %f \n", ++c);
    printf("--d = %f \n", --d);
    return 0;
}
```

Compiled by Ms. Prajakta Joshi
#include <stdio.h>
int main()
{
    int a;
    float b;
    double c;
    char d;
    printf("Size of int=\%lu bytes\n", sizeof(a));
    printf("Size of float=\%lu bytes\n", sizeof(b));
    printf("Size of double=\%lu bytes\n", sizeof(c));
    printf("Size of char=\%lu byte\n", sizeof(d));
    return 0;
}
## RELATIONAL OPERATORS

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>==</code></td>
<td>Checks if the values of two operands are equal or not. If yes, then the condition becomes true.</td>
<td><code>(A == B) is not true.</code></td>
</tr>
<tr>
<td><code>!=</code></td>
<td>Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.</td>
<td><code>(A != B) is true.</code></td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.</td>
<td><code>(A &gt; B) is not true.</code></td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.</td>
<td><code>(A &lt; B) is true.</code></td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.</td>
<td><code>(A &gt;= B) is not true.</code></td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.</td>
<td><code>(A &lt;= B) is true.</code></td>
</tr>
</tbody>
</table>

Compiled by Ms. Prajakta Joshi
```c
#include <stdio.h>
main()
{
int a = 21; int b = 10; int c;
if( a == b )
{
    printf("Line 1 - a is equal to b\n");
}
else
{
    printf("Line 1 - a is not equal to b\n");
}
}```
## LOGICAL OPERATORS

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.</td>
<td>(A &amp;&amp; B) is false.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will return false.</td>
<td>!(A &amp;&amp; B) is true.</td>
</tr>
</tbody>
</table>
• #include <stdio.h>
• main() {
• int a = 5; int b = 20; int c ;
• if ( a && b )
• {
• printf("Line 1 - Condition is true\n" );
• }
• if ( a || b )
• {
• printf("Line 2 - Condition is true\n" ); }
/* lets change the value of a and b */
a = 0; b = 10;
if ( a && b )
{
    printf("Line 3 - Condition is true\n" );
}
else
{
    printf("Line 3 - Condition is not true\n" );
}
if ( !(a && b) )
{
    printf("Line 4 - Condition is true\n" );
}
# ASSIGNMENT OPERATORS

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Simple assignment operator. Assigns values from right side operands to left side operand</td>
<td>C = A + B will assign the value of A + B to C</td>
</tr>
<tr>
<td>+=</td>
<td>Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.</td>
<td>C += A is equivalent to C = C + A</td>
</tr>
<tr>
<td>-=</td>
<td>Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.</td>
<td>C -= A is equivalent to C = C - A</td>
</tr>
<tr>
<td>*=</td>
<td>Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.</td>
<td>C *= A is equivalent to C = C * A</td>
</tr>
<tr>
<td>/=</td>
<td>Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.</td>
<td>C /= A is equivalent to C = C / A</td>
</tr>
<tr>
<td>%=</td>
<td>Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.</td>
<td>C %= A is equivalent to C = C % A</td>
</tr>
<tr>
<td>&lt;&lt;=</td>
<td>Left shift AND assignment operator.</td>
<td>C &lt;&lt;= 2 is same as C = C &lt;&lt; 2</td>
</tr>
<tr>
<td>&gt;&gt;=</td>
<td>Right shift AND assignment operator.</td>
<td>C &gt;&gt;= 2 is same as C = C &gt;&gt; 2</td>
</tr>
<tr>
<td>&amp;=</td>
<td>Bitwise AND assignment operator.</td>
<td>C &amp;= 2 is same as C = C &amp; 2</td>
</tr>
<tr>
<td>^=</td>
<td>Bitwise exclusive OR and assignment operator.</td>
<td>C ^= 2 is same as C = C ^ 2</td>
</tr>
<tr>
<td></td>
<td>=</td>
<td>Bitwise inclusive OR and assignment operator.</td>
</tr>
</tbody>
</table>

Compiled by Ms. Prajakta Joshi
• #include <stdio.h>
• main()
  • {
  •   int a = 21; int c ;
  •   c = a;
  •   printf("Line 1 - = Operator Example, Value of c = %d\n", c);
  •   c += a;
  •   printf("Line 2 - += Operator Example, Value of c = %d\n", c);
  •   c -= a;
  •   printf("Line 3 - -= Operator Example, Value of c = %d\n", c);
  •   c *= a;
  •   printf("Line 4 - *= Operator Example, Value of c = %d\n", c);
  • }
• c = 200; c %= a;
• printf("Line 6 - %= Operator Example, Value of c = %d\n", c);
• c <<= 2;
• printf("Line 7 - <<= Operator Example, Value of c = %d\n", c);
• c >>= 2;
• printf("Line 8 - >>= Operator Example, Value of c = %d\n", c);
• c &= 2;
• printf("Line 9 - &= Operator Example, Value of c = %d\n", c);
• c ^= 2;
• printf("Line 10 - ^= Operator Example, Value of c = %d\n", c);
• c |= 2;
• printf("Line 11 - |= Operator Example, Value of c = %d\n", c);
• }

Compiled by Ms. Prajakta Joshi
• Conditional operators return one value if condition is true and returns another value if condition is false.
• This operator is also called as ternary operator.
• Syntax : (Condition? true_value: false_value);
• Example : (A > 100 ? 0 : 1);
```c
#include <stdio.h>

int main()
{
    int x=1, y;
    y = ( x == 1 ? 2 : 0 );
    printf("x value is %d\n", x);
    printf("y value is %d", y);
}
```
#include<stdio.h>

int main()
{
  int age;

  printf(" Please Enter your age here: \n ");
  scanf(" %d ", &age);

  (age >= 18) ? printf(" You are eligible to Vote ") :
                 printf(" You are not eligible to Vote ");

  return 0;
}
TYPE CONVERSION/TYPEx CASTING

• Implicit Type Conversion/ Casting
• Explicit Type Conversion/ Casting
• Implicit type casting means conversion of data types without losing its original meaning.
• This type of typecasting is essential when you want to change data types **without** changing the significance of the values stored inside the variable.
• Implicit type conversion happens automatically when a value is copied to its compatible data type.
• During conversion, strict rules for type conversion are applied.
• If the operands are of two different data types, then an operand having lower data type is automatically converted into a higher data type.
```c
#include<stdio.h>
int main()
{
    short a=10; //initializing variable of short data type
    int b; //declaring int variable
    b=a; //implicit type casting
    printf("%d\n",a);
    printf("%d\n",b);
}
```
EXAMPLE OF ITC FROM CHAR TO INT

```c
#include<stdio.h>
int main()
{
    int x = 10;    // integer x
    char y = 'a';  // character c

    // y implicitly converted to int. ASCII
    // value of 'a' is 97
    x = x + y;

    // x is implicitly converted to float
    float z = x + 1.0;

    printf("x = %d, z = %f", x, z);
    return 0;
}
```

Compiled by Ms. Prajakta Joshi
EXPLICIT TYPE CONVERSION

• When interpretation is between a variable having a data type with respect to size & type both, this conversion is not possible for compiler automatically.
• It is performed by the programmer.
• In this type casting programmer tells compiler to type cast one data type to another data type using type casting operator.
• but there is some risk of information loss is there, so one needs to be careful while doing it.
C PROGRAM TO DEMONSTRATE EXPLICIT TYPE CASTING

• #include<stdio.h>

• int main()
• {
  •    double x = 1.2;
  •    // Explicit conversion from double to int
  •    int sum = (int)x + 1;
  •
  •    printf("sum = %d", sum);
  •    return 0;
  •}

Compiled by Ms. Prajakta Joshi
C PROGRAM TO DEMONSTRATE EXPLICIT TYPE CASTING

• `#include<stdio.h>`
• `int main()`
• `{`
  • `float a = 1.2;`
  • `//int b = a; //Compiler will throw an error for this`
  • `int b = (int)a + 1;`
  • `printf("Value of a is %f\n", a);`
  • `printf("Value of b is %d\n", b);`
  • `return 0;`
• `}`

Compiled by Ms. Prajakta Joshi
INPUT & OUTPUT FUNCTIONS

Console Input/Output functions

Formatted functions

<table>
<thead>
<tr>
<th>Type</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>scanf()</td>
<td>printf()</td>
</tr>
<tr>
<td>int</td>
<td>scanf()</td>
<td>printf()</td>
</tr>
<tr>
<td>float</td>
<td>scanf()</td>
<td>printf()</td>
</tr>
<tr>
<td>string</td>
<td>scanf()</td>
<td>printf()</td>
</tr>
</tbody>
</table>

Unformatted functions

<table>
<thead>
<tr>
<th>Type</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>getch()</td>
<td>putch()</td>
</tr>
<tr>
<td></td>
<td>getche()</td>
<td>putchar()</td>
</tr>
<tr>
<td>int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>float</td>
<td></td>
<td></td>
</tr>
<tr>
<td>string</td>
<td>gets()</td>
<td>puts()</td>
</tr>
</tbody>
</table>
GETCH() FUNCTION

• The getch() function reads the alphanumericonic character input from the user. But, that the entered character will not be displayed.

```c
#include <stdio.h>
#include <conio.h>

int main() {
    printf("\nHello, press any alphanumeronic character to exit ");
    getch();
    return 0; }
```

Compiled by Ms. Prajakta Joshi
GETCHE() FUNCTION

- getche() function reads the alphanumerical character from the user input. Here, character you entered will be echoed to the user until he/she presses any key.

```c
#include <stdio.h>  //header file section
#include <conio.h>
int main() {
    printf("\nHello, press any alphanumerical character or symbol to exit \n ");
    getche();
    return 0; }
```
GETCHAR() FUNCTION

• The getchar() function reads character type data form the input.
• The getchar() function reads one character at a time till the user presses the enter key.
#include <stdio.h> //header file section
#include <conio.h>
  int main()
  {
    char c;
    printf("Enter a character : ");
    c = getchar();
    printf("\nEntered character : %c ", c);
    return 0;
  }
The `gets()` function can read a full string even blank spaces presents in a string.
But, the `scanf()` function leave a string after blank space is detected.
The `gets()` function is used to get any string from the user.
GETS() FUNCTION

#include <stdio.h>
#include <conio.h>

int main()
{
    char c[25];
    printf("Enter a string : ");
    gets(c);
    gets(c);
    printf("\n%s is awesome ",c);
    return 0;
}
PUTCH() FUNCTION

- The putch() function prints any alphanumerical character.

```c
#include <stdio.h> //header file section
#include <conio.h>

int main() {
    char c;
    printf("Press any key to continue\n ");
    c = getch();
    printf("input : ");
    putch(c);
    return 0; }
```
• putchar() function prints only one character at a time.

```c
#include <stdio.h> //header file section
#include <conio.h>

int main() {
    char c = 'K';
    putchar(c);
    return 0;
}
```
PUTS() FUNCTION

- The puts() function prints the character array or string on the console. The puts() function is similar to printf() function, but we cannot print other than characters using puts() function.
#include <stdio.h>
#include <conio.h>

int main()
{
    char c[25];
    printf("Enter your Name : ");
    gets(c);
    puts(c);
    puts(c);
    return 0;
}