forbeginners.c tutorials.out

CSE1062 | CCS1063 ‘Practicals’ {

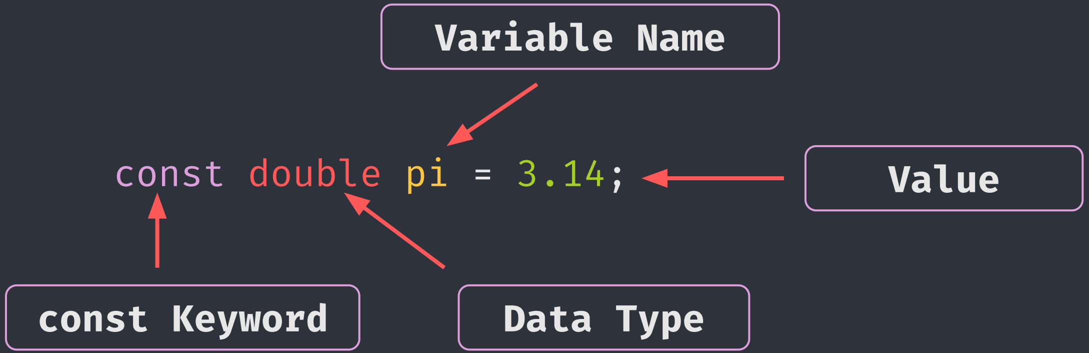
[Fundamentals of Computer Programming]

< Tutorial Session 02 - Operators >

}

Constant variables

The qualifier const can be applied to the declaration of any variable to specify that its value will not be changed.



```c
const double pi = 3.14;
```

Preprocessor constants

Named constants may also be created using a preprocessor . Preprocessor is an editor , it perform search and replace task.

```
#define <VAR_NAME> <VALUE>
```

<VAR_NAME> is a placeholder for the name of the constant.

It's recommended that you name constants in the uppercase, as it helps differentiate them from other variables defined in the program.

<VALUE> is a placeholder for the value that <VAR_NAME> takes.

#define is a preprocessor directive.

Before the compilation of the program, the preprocessor replaces all occurrences of <VAR_NAME> with <VALUE>.

Preprocessor constants

```c
#include <stdio.h>
#define STUDENT_ID 27
#define COURSE_CODE 502

void main()
{
    printf("Student ID: %d is taking the class %d\n",
STUDENT_ID, COURSE_CODE);
}
```

Type conversion

It can Temporarily change the type of the variable in C.

There are two types of type conversion.

1. Implicit Conversion

   the value of one type is automatically converted to the value of another type.

2. Explicit Conversion

   Need to convert values of one data type to another type manually by type casting.

Implicit Conversion

```c
#include <stdio.h>

void main() {

    int  i = 17;
    char c = 'c'; /* ascii value is 99 */
    int sum;

    sum = i + c;
    printf("Value of sum : %d\n", sum );
}
```

Explicit Conversion

```c
#include <stdio.h>

void main() {

    int sum = 17, count = 5;
    double mean;

    mean = (double) sum / count;
    printf("Value of mean : %f\n", mean );
}
```

C Operators

< An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. >

Arithmetic Operators

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

https://www.tutorialspoint.com/cprogramming/c_operators.htm

Arithmetic Operators

```c
#include <stdio.h>
void main()
{
    int a = 9,b = 4, c;
    c = a+b;
    printf("a+b = %d \n",c);
    c = a-b;
    printf("a-b = %d \n",c);
    c = a*b;
    printf("a*b = %d \n",c);
    c = a/b;
    printf("a/b = %d \n",c);
    c = a%b;
    printf("Remainder of a/b = %d \n",c);
    }
```

Assignment Operators

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A

https://www.tutorialspoint.com/cprogramming/c_operators.htm

Assignment Operators…

/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

https://www.tutorialspoint.com/cprogramming/c_operators.htm

Assignment Operators…

```c
#include <stdio.h>
void main()
{
    int a = 5, c;
    c = a;      // c is 5
    printf("c = %d\n", c);
    c += a;     // c is 10
    printf("c = %d\n", c);
    c -= a;     // c is 5
    printf("c = %d\n", c);
    c *= a;     // c is 25
    printf("c = %d\n", c);
    c /= a;     // c is 5
    printf("c = %d\n", c);
    c %= a;     // c = 0
    printf("c = %d\n", c);
    }
```

Relational Operators

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is true.

Relational Operators…

>=	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.	(A >= B) is not true.
<=	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.	(A <= B) is true.

Relational Operators…

```c
#include <stdio.h>
void main()
{
    int a = 5, b = 5, c = 10;

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

Logical Operators

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.
!	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 make it false.	!(A && B) is true.

AND Operator

Logical AND. True only if all operands are true

Value 01	Value 02	Result
True	True	True
True	False	False
False	True	False
False	False	False

Or Operator

Logical OR. True only if either one operand is true

Value 01	Value 02	Result
True	True	True
True	False	True
False	True	True
False	False	False

Not Operator

Logical NOT. True only if the operand is 0

Value 01	Result
True	False
False	True

Logical Operators

```c
#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10, result;

    result = (a == b) && (c > b);
    printf("(a == b) && (c > b) is %d \n", result);

    result = (a == b) || (c < b);
    printf("(a == b) || (c < b) is %d \n", result);

    result = (a != b) || (c < b);
    printf("(a != b) || (c < b) is %d \n", result);

    result = !(a != b);
    printf("!(a != b) is %d \n", result);

    result = !(a == b);
    printf("!(a == b) is %d \n", result);
}
```

Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation.

Assume A = 60 and B = 13 in binary format, they will be as follows - A = 0011 1100 B = 0000 1101

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001

Bitwise Operators…

A = 0011 1100   B = 0000 1101

~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = ~(60), i.e,. -0111101
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111

Bitwise Operators…

```c
#include <stdio.h>

void main() {

   unsigned int a = 60; /* 60 = 0011 1100 */
   unsigned int b = 13; /* 13 = 0000 1101 */
   int c = 0;

   c = a & b;       /* 12 = 0000 1100 */
   printf("Line 1 - Value of c is %d\n", c );
   c = a | b;       /* 61 = 0011 1101 */
   printf("Line 2 - Value of c is %d\n", c );
   c = a ^ b;       /* 49 = 0011 0001 */
   printf("Line 3 - Value of c is %d\n", c );
   c = ~a;          /*-61 = 1100 0011 */
   printf("Line 4 - Value of c is %d\n", c );
   c = a << 2;     /* 240 = 1111 0000 */
   printf("Line 5 - Value of c is %d\n", c );
   c = a >> 2;     /* 15 = 0000 1111 */
   printf("Line 6 - Value of c is %d\n", c );
}
```

sizeof and ternary operators

Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
? :	Conditional Expression.	If Condition is true ? then value X : otherwise value Y

sizeof and ternary operators

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

   int a = 4;
   short b;
   double c;
   int* ptr;
   /* example of sizeof operator */
   printf("Line 1 - Size of variable a = %d\n", sizeof(a) );
   printf("Line 2 - Size of variable b = %d\n", sizeof(b) );
   printf("Line 3 - Size of variable c= %d\n", sizeof(c) );
   /* example of & and * operators */
   ptr = &a;    /* 'ptr' now contains the address of 'a'*/
   printf("value of a is  %d\n", a);
   printf("*ptr is %d.\n", *ptr);
   printf("ptr is %p.\n", ptr);
   /* example of ternary operator */
   a = 10;
   b = (a == 1) ? 20: 30;
   printf( "Value of b is %d\n", b );
   b = (a == 10) ? 20: 30;
   printf( "Value of b is %d\n", b );
}
```

Pre/Post Increment Operators

< There are two types of the increment operators >

- **Pre-increment operator ++variableName**
  - The pre-increment operator is used to increment the value of an operand by 1 before using it in the mathematical expression.
  - In other words, the value of a variable is first incremented, and then the updated value is used in the expression.

- **Post-increment operator variableName++**
  - It increments the value of the operand by 1 after using it in the mathematical expression.
  - In other words, the variable's original value is used in the expression first, and then the post-increment operator updates the operand value by 1.

Pre/Post Increment Operators

```c
#include <stdio.h>

int main() {
    int num = 5;
    int sum_pre = 0;
    int sum_post = 0;

    // Pre-increment: The value is incremented before being used
    sum_pre = (++num) + (++num) + (++num);

    // Post-increment: The value is incremented after being used
    sum_post = (num++) + (num++) + (num++);

    printf("Sum using pre-increment: %d\n", sum_pre);
    printf("Sum using post-increment: %d\n", sum_post);

    return 0;
}
```

Getting input from Keyboard

```c
#include <stdio.h>

void main(){
    int a; char c; float nu;
    printf("input a number:");
    scanf("%d",&a);
    printf("%d\ninput a character:",a);
    scanf(" %c",&c);  // Add a space before %c to consume the newline character
    printf("%c\ninput a float:",c);
    scanf("%f",&nu);
    printf("%f",nu);
}
```

Why we use ‘&’ sign

scanf() expects pointers (the address to the variable) as arguments.

some types - like arrays (eg. int arr[6]), strings (eg. char name[20]) and pointers(eg. char* address) are already addresses and you can pass them to scanf by simply using their names.

```
char name[20];   scanf("%s",name);
```

```
Thanks; {
    ‘Do you have any questions?’

        < bgamage@sjp.ac.lk >

}
```