Statements, Variables and Operators

Statements

A statement is a **command** given to the computer that **instructs** the computer to take a **specific action**, such as **display** to the **screen**, **collect input** etc.

- A computer program is made up of a series of **statements**
- **Simple statements** do not contain other statements
  - x = 2;
- **Compound statements** have other statements inside them
  - **Conditionals**: if(happy) { smile(); }
  - **Loops**: while((c = getchar()) != EOF) { putchar(c); }

Types of Statements

- Statement: Calling functions
  - printf(“Hello World”);
- Statement: Assignment statement
  - Required when a **value** is needed to copy to a **variable**
  - **Idea**: Variable (x) ← Value (y)
  - **Values** in C are derived from **Expressions.**
- Expressions:
  - **Combination** of **variables** or **constants** and **operators**
  - An **Expression** always **yields** a **single** value

Examples of Expressions

Algebraic Expression	C Expression
1+32	1+32
43 × 9.98 – 3.5 ÷ 4	43 * 9.98 – 3.5 / 4
ab + bc + ca	a*b + b*c + c*a
(m+n)(mn)	(m+n)*(m*n)
$5x^2 + 4x + 1$	5*x*x + 4*x + 1
$\left(x - \frac{10+\frac{y}{b}}{z}\right)\left(p + \frac{q}{1-\frac{m}{n}}\right)$	$(x - ((10 + (y/b))/z))(p + (q/(1 - (m/n))))$

C Comments

A comment is a programming language construct used to attach **programmer-readable annotations (explanations)** in the source code of a computer program.

- This helps others **understand** the program
- Also helps the programmer **understand** fully after **a long time**
- C comments **starts with a /*** and **ends with a */**
  - For example: /* this is a comment
    written in two lines */

The **compiler** simply **skips** the **comments** before **compilation**!
So we don't need to worry about the grammar of the comments.

Keywords

Keywords are **predefined**, **reserved words** used in programming that have **special meanings** to the compiler.

Examples:

auto, double, int, break, else, long, switch, case, enum, register, typedef, char, extern, return, union, continue, for, signed, void, do, if, static, while, default, goto, sizeof, volatile, const, float, short, unsigned etc

Identifiers

- Identifier refers to name given to **entities** such as
  - Variables
  - Functions
  - Structures etc.
- Identifiers must be **unique**.
- Naming Format:
  - Can have **letters** (both uppercase and lowercase letters), **digits** and **underscores**.
  - The **first letter** of an identifier should be either a **letter** or an **underscore**
  - **You cannot use keywords as identifiers**
  - Although maximum length is not defined, the identifier, longer than 31 characters should not be used

Variables

- The variable is the **container** to hold data.
- It is a way to represent **memory location** through symbol
- Its value can be **changed**
- It can be **reused** many times
- Naming format follows **identifiers**

Variables that are **not allowed to change** value are **constants**.

Data used for representing **fixed** values are **literals**

Variables

Data used for representing **fixed** values are **literals**

Different Literals:

- Integer Literal [1, 2, 5, 10]
- Floating Point Literals [0.1, 2.16, 5.9, 3.3]
- Characters [‘a’, ‘b’, ‘1’, ‘#’]
- Strings [“”, “string”, “pro”, “123”, “1c4x”]
- Escape Sequences [\n, \b, \t, \\ ]

Character Codes

- **ASCII** or American Standard Code for Information Interchange
- 7-bit character code representing unique characters
- Extended ASCII is 8-bit character code
- In C, a character variable contain the ascii value of a character variable

Storage Types

Storage Class	Declaration	Storage	Default Initial Value	Scope	Lifetime
auto	Inside a function/block	Memory	Unpredictable	Within the function/block	Within the function/block
register	Inside a function/block	CPU Registers	Garbage	Within the function/block	Within the function/block
extern	Outside all functions	Memory	Zero	Entire the file and other files where the variable is declared as extern	program runtime
Static (local)	Inside a function/block	Memory	Zero	Within the function/block	program runtime
Static (global)	Outside all functions	Memory	Zero	Global	program runtime

C Data Types

- Basic Types:
  - int (4 bytes)
  - float (4 bytes)
  - double (8 bytes)
  - char (1 byte)
  - unsigned int (4 bytes)
  - unsigned double (8 bytes)
  - short int (2 bytes)
  - long int (4 bytes)
  - long long int (8 bytes)

- Derived Data Type:
  - Structure
  - Array
  - Pointer
  - Function

Operator

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

- Arithmetic Operators [+ - * / %]
- Relational Operators [> < == <= >= !=]
- Logical Operators [&& || !]
- Assignment operators [= += -= *= /= %= <<= >>= &= |= ^=]
- Increment / Decrement operator [++ --]
- Conditional Operator [?:]
- Bitwise Operator [& | ^ ~ << >>]
- Special Operator [* & sizeof()]

Arithmetic Operator

Operator	Meaning	Unary Example	Binary Example
+	Addition (Binary & Unary)	+5 +x	5 + x + 10 10 + y
-	Subtraction (Binary & Unary)	-10 -y	5 - y 15 - y - z
*	Multiplication	Not Available	10 * y 55 * z * 33
/	Division	Not Available	155 / z 255 / z / k
%	Remainder	Not Available	155 % 30 % z 155 % 2

Relational Operator (x=14, y=4)

Operator	Meaning	Expression	Value
==	Equal to	5 == 5 X == y - 1	1 0
!=	Not Equal to	0 != 1 X != y	1 1
>=	Greater than or Equal to	10 >= 3 3 >= 10	1 0
<=	Less than or Equal to	10 <= 10 9 <= 10	1 1
>	Greater than	9 > 10 10 > 10	0 0
<	Less than	9 < 10 10 < 9	1 0

Logical Operator

<table>
<tr><th>Operator</th><th>Meaning</th><th>Truth Table</th></tr>
<tr><td>&&</td><td>Logical AND</td><td>
<table>
<tr><th>A</th><th>B</th><th>A && B</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td></tr>
<tr><td>||</td><td>Logical OR</td><td>
<table>
<tr><th>A</th><th>B</th><th>A || B</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td></tr>
<tr><td>!</td><td>Logical NOT</td><td>
<table>
<tr><th>A</th><th>!A</th></tr>
<tr><td>0</td><td>1</td></tr>
<tr><td>1</td><td>0</td></tr>
</table>
</td></tr>
</table>

Operator Precedence

Precedence	Operator	Meaning of operator	Associativity
1	() [] -> .	Functional call Array element reference Indirect member selection Direct member selection	Left to right
2	! ~ + - ++ -- & * sizeof (type)	Logical negation Bitwise(1 's) complement Unary plus Unary minus Increment Decrement Dereference (Address) Pointer reference Returns the size of an object Typecast (conversion)	Right to left

Operator Precedence

Precedence	Operator	Meaning of operator	Associativity
3	* / %	Multiply Divide Remainder	Left to right
4	+ -	Binary plus(Addition) Binary minus(subtraction)	Left to right
5	<< >>	Left shift Right shift	Left to right
6	< <= > >=	Less than Less than or equal Greater than Greater than or equal	Left to right

Operator Precedence

Precedence	Operator	Meaning of operator	Associativity
7	== !=	Equal to Not equal to	Left to right
8	&	Bitwise AND	Left to right
9	^	Bitwise exclusive OR	Left to right
10	|	Bitwise OR	Left to right
11	&&	Logical AND	Left to right
12	||	Logical OR	Left to right
13	?:	Conditional Operator	Right to left

Operator Precedence

Precedence	Operator	Meaning of operator	Associativity
14	= *= /= %= += -= &= ^= |= <<= >>=	Simple assignment Assign product Assign quotient Assign remainder Assign sum Assign difference Assign bitwise AND Assign bitwise XOR Assign bitwise OR Assign left shift Assign right shift	Right to left
15	,	Separator of expressions	Left to right

Thank You