BITS Pilani
Pilani Campus

Module 5 – Data types, variables, operators & expressions

Dr. Jagat Sesh Challa
Department of Computer Science & Information Systems

Module Overview

- **Program Data and Variables**
- **Data Types**
- **Constants**
- **Type Conversions**
- **Operators and Expressions**

innovate achieve lead

BITS Pilani
Pilani Campus

Program Data and Variables

Program Data

Programs deal with data!

Examples:

- Name
- Weight
- Quantity
- Price

Program Data

Program data occurs in the form of

- *Variables*
- *Constants*

Variables

- A name given to a memory location
- The name used to access a variable is also known as an identifier
- C has strict rules for variable naming
- Variable name can begin with a *letter* or *underscore*, and comprise of *letters, digits* or *underscore*. Names are case-sensitive
- Certain reserved keywords cannot be used as variable names (`continue`, `if`, `short`, `union`, `return` …)
- Use meaningful names!

Variables (contd.)

- A variable *must* be declared before its use

  Examples:

  ```
  int max;   /* declares a variable max that can store integer values */
  int age, quantity, price;
  /* declares three variables that can store integer values */
  ```

- To initialize a variable, use =

  Example:

  ```
  age = 21;
  ```

- Declaration and initialization can be combined in one statement

  Example:

  ```
  int age = 4, quantity = 500, price = 999;
  ```

- An uninitialized variable is simply *junk* (in general)

Variables in Main Memory

- Consider the following C Program:

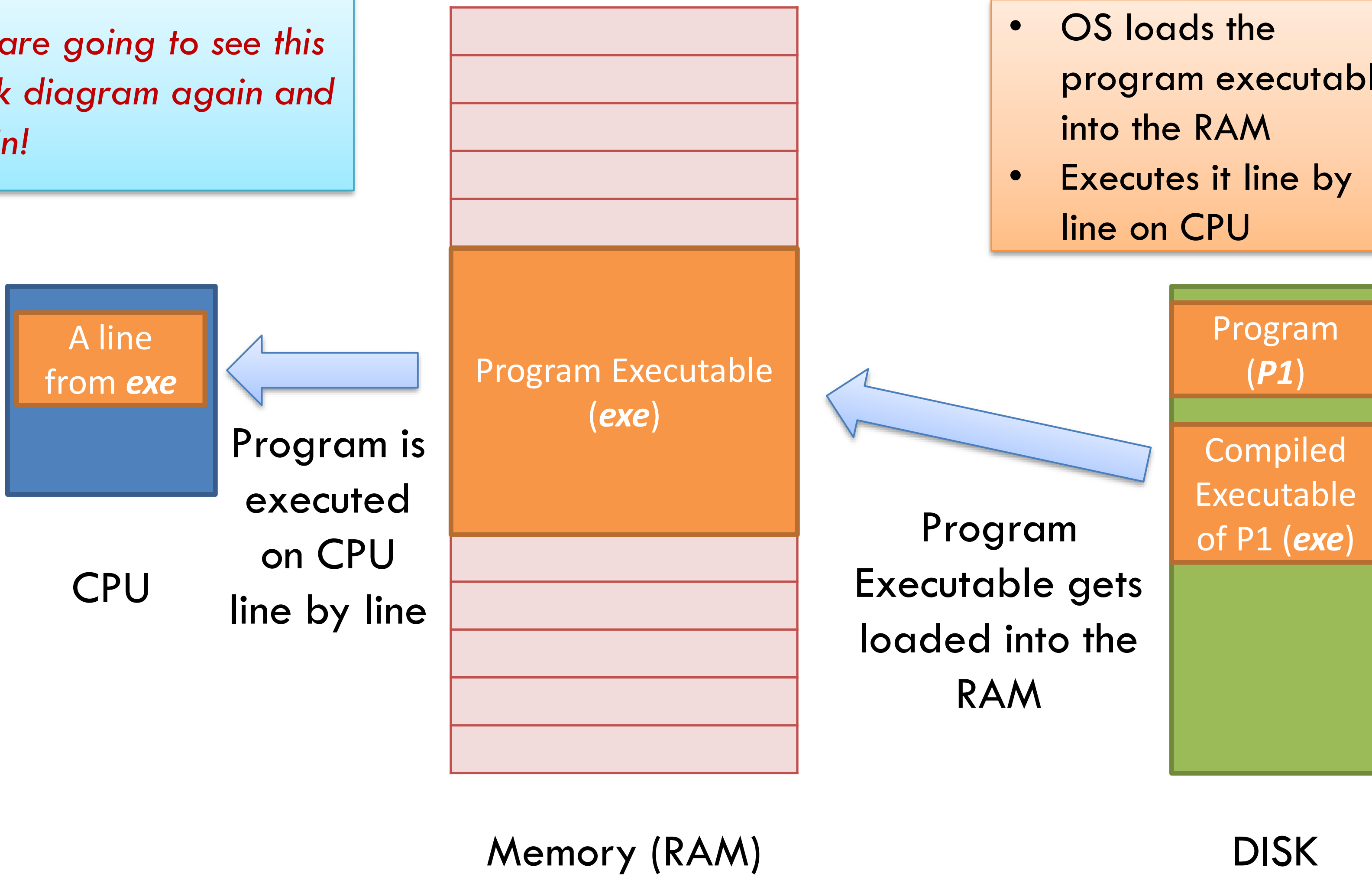
```
#include <stdio.h>
int main()
{
   int num1, num2, num3;
   num1 = 2;
   num2 = 4;
   num3 = num1 + num2; // computing the sum of num1 and num2
   printf("The sum is: %d \n", num3); // printing the sum
   return 0;
}
```

- This program has three variables: `num1`, `num2` and `num3`.
- *Where are these variables stored?*

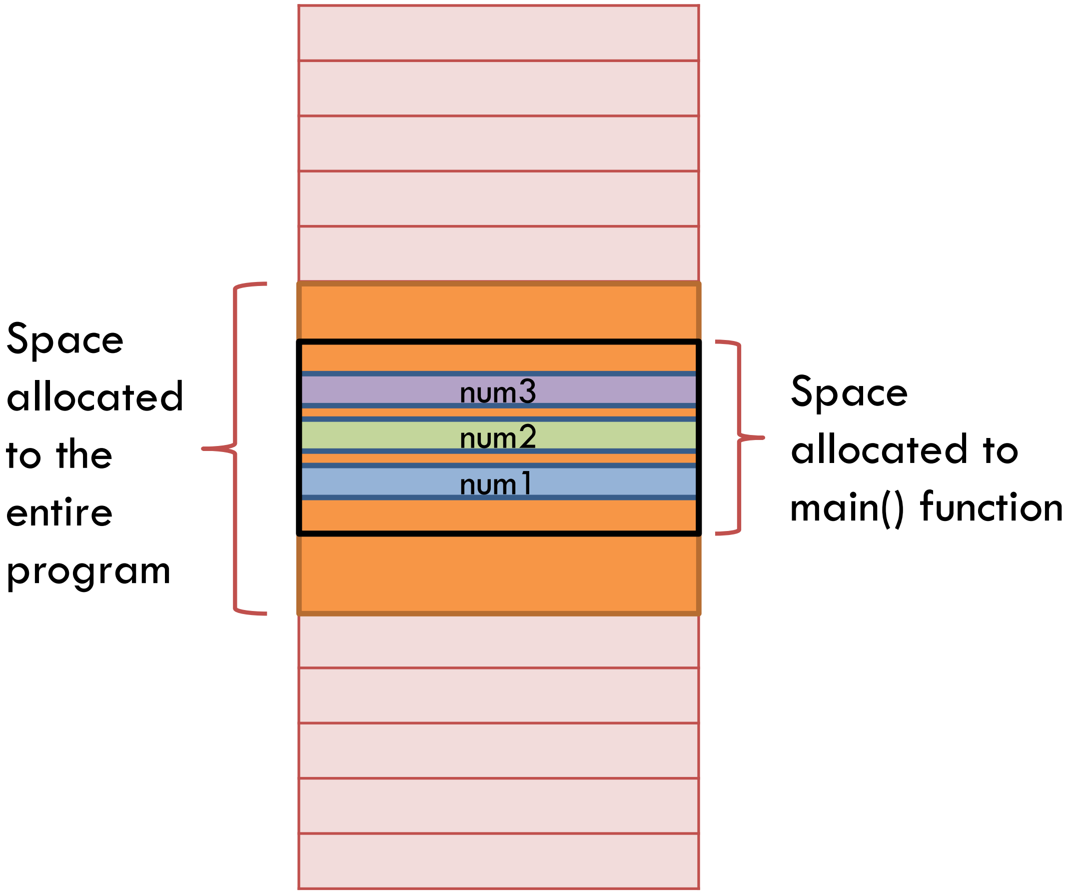
Remember this block diagram!

We are going to see this block diagram again and again!

- OS loads the program executable into the RAM
- Executes it line by line on CPU



Look at the main memory only



Memory (RAM)

- When we compile and execute a program, OS allocates some space in the main memory
- The declared variables are stored in that allocated space.
- In our example, `num1`, `num2` and `num3` are stored in this space.
- More specifically in the space allocated to the `main()` function, within the above space.
    - We will study about memory allocation and functions in greater detail later!

innovate achieve lead

BITS Pilani
Pilani Campus

Data Types

Data types

- C is a *typed* language
- Every data item has a type associated with it.
- Examples of declaring variables:
  - `int num1;`
  - `short num2;`
  - `long num3;`
  - `float cgpa;`
  - `double percentage;`
  - `long double pqr;`
  - `char c1;`

Fundamental data types in C

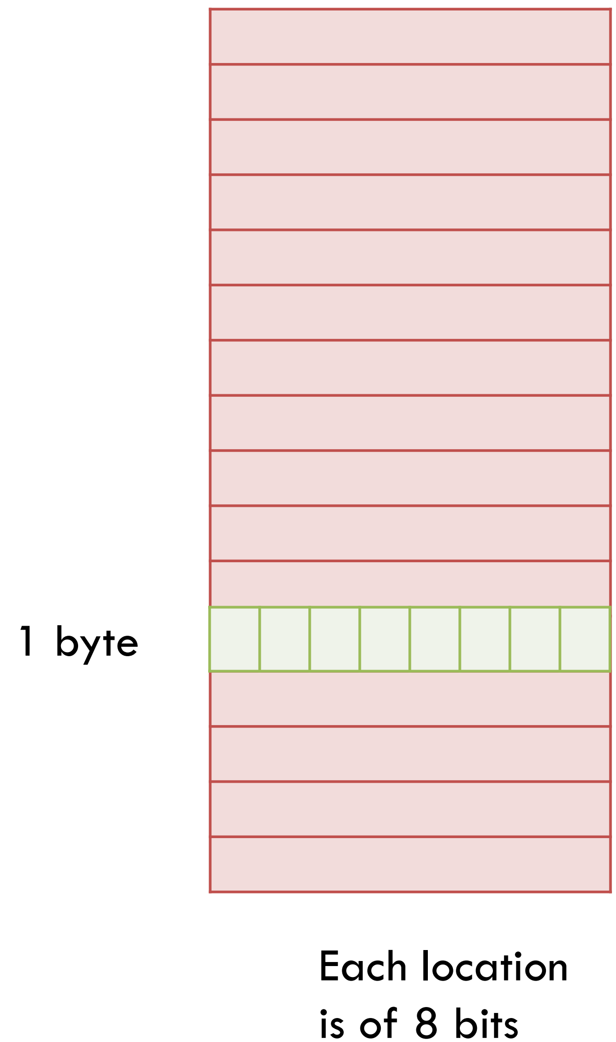
- Integer (`short`, `int`, `long`, `long long`)
- Floating point (`float`, `double`, `long double`)
- Character (`char`)
- Fixed size of each data type / sub-type.

Machine Readable size of variables

- Every variable occupies space in the program allocated space of the main memory
- *How much space does each variable occupy?*
- It depends on its type!
- For example, variable of type `int` occupies either *2 or 4 bytes* depending upon the specific compiler you are using.
- *What is a* ***byte****?*
- *Before we answer this question, let us see how the main memory is organized!*

Organization of Main Memory

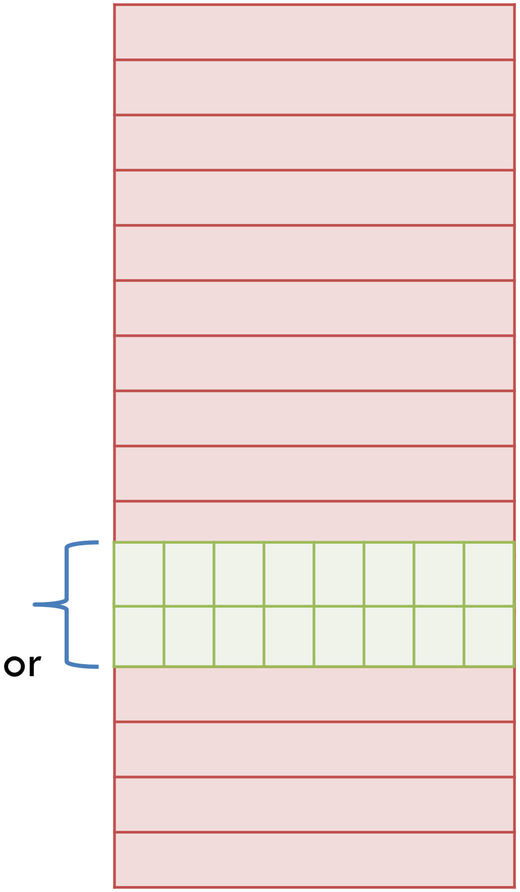
- Main memory is typically a table of memory locations of 8 bits (0/1) each
- A set of contiguous 8 bits is known as a byte
- So, each location in this main memory is of one byte
  - We call byte-addressable memory
- Each location is associated with an address.



Storing int in main memory

- Remember how integer values are represented in 2's complement representation.
- Our machines use 2's complement representation to store integer variables.
- If int variables are of 2 bytes size (or 16 bits), then each int variable shall occupies 2 contiguous locations in the memory

Space occupied by int variable, which is 2 bytes or 16 bits



Types of integer variables

Type / Subtype	Minimum size (bytes)	Range	Format Specifier
short	2	-32,768 to 32,767 (**-2^{15} to $2^{15}-1$**)	%hd
unsigned short	2	0 to 65,535 (**0 to $2^{16}-1$**)	%hu
int	2 or 4	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647	%d
unsigned int	2 or 4	0 to 65,535 or 0 to 4,294,967,295	%u
long	4 or 8	-2,147,483,648 to 2,147,483,647 or -9223372036854775808 to 9223372036854775807	%ld
unsinged long	4 or 8	0 to 4,294,967,295 or 0 to 18446744073709551615	%lu
long long	8	-9223372036854775808 to 9223372036854775807	%lld
unsinged long long	8	0 to 18446744073709551615	%llu

Knowing the size of data types

- How do we know the size of a data type for our C compiler?
- Use `sizeof()` operator
- Example:

```
printf("Size of int is %lu bytes", sizeof(int));
// prints size of int in bytes. The format specifier is unsigned long int (%lu) on gcc.
```

- Note: By default, integer data type is *signed*.

Types of floating point numbers

While integers classified by size, floating point numbers are classified by *precision*

Type / Subtype	min precision (digits)	min size (bytes)	mantissa-exponent	Format Specifier
float	6	4	23-8	%f, %e
double	10	8	52-11	%lf
long double	10	8-16	112-15	%Lf

Questions for you

What happens when you store a very large value in an 'int'?

What happens when you use a wrong format specifier?

What happens when you store a large 'double' value in a 'float'?

What is the size of 'long long' on your machine?

Character type

- If everything is in binary, how do we deal with characters?
- Use Character-encoding schemes, such as ASCII
- ASCII defines an encoding for representing English characters as numbers, with each letter assigned a number from 0 to 127.
- 8-bit ASCII used, so char data type is one byte wide

ASCII Encoding

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Computations with char

```
char ch1 = 'A', ch2;
printf("%c\n", ch1);
ch2 = ch1 + 1;
printf("%c\n", ch2);
```

Non-printable chars

- Escape sequences – backslash followed by the lowercase letter
  - Examples: `\n`, `\t`, etc.

- Printing special characters like `'`, `"`, `\`, etc.
  - use a preceding `\`

Chars and Strings

- “Internet” is a string, which is basically a bunch of characters put together
- A string in C is an *array* of chars.
- *We will study arrays and strings in greater detail later!*
- Note the use of double quotes
- What is the distinction between ‘A’, “A”, ‘Axe’ and “Axe” ??

Constants

Declaring Constants

2 ways:

1. Using `const` qualifier

```
const float pi = 3.14f; //pi is read-only
```

2. Symbolic constants using `#define`

```
#define PI 3.14f  //no semi-colon used
```

What is the difference between these two?

Constants

- C specifies a default type for a constant.
- We can also force a type.
- Default for an integer is int. If a constant doesn't fit in int, the next higher type is tried.
- Suffixes can be used to coerce the data type.
- U or u for unsigned. L or l for long….
- For a floating point number, the default is double.
- Suffix F or f demotes it to float; L or l promotes it to long double.
- A character constant is stored as an int (!!)

Declaring Constants

- Using `#define`
- Syntax: `#define variable_name value`
- Note: no use of semicolon at the end
- Example:

```
#include<stdio.h>
#define X 10  //always written before int main()
int main(){
    int A = X;
    X = X+10; //error: lvalue required as left operand of assignment
    return 0;
}
```

Constants vs Variables

Constants	Variables
A constant does not change its value over time.	A variable, on the other hand, changes its value dependent on the equation.
Value once assigned can't be altered by the program.	Values can be altered by the program.

Type Conversions

Type Conversions

Implicit

- If either operand is `long double`, convert the other into `long double`.
- Otherwise, if either operand is `double`, convert the other into `double`.
- Otherwise, if either operand is `float`, convert the other into `float`.
- Otherwise, convert `char` and `short` to `int`.
- Then, if either operand is `long`, convert the other to `long`.

Explicit (also known as coercion or typecasting)

Implicit Type Conversion

e.g.,

```
float A = 100/3;    output: 33.000000
float A = 100/3.0  output: 33.333333

int A = 10;
A = A + 'B';    Output: 76
  ‘B’ is type converted to integer

A = ‘A’ + ‘B’;  Output ?
float X = 0.2; double A = X/0.1;  Output ?
```

Explicit Type Conversion (Typecasting)

Example1:

Conversion of integer to a float variable

```
int a = 10;
float f = (float) a;
```

Example2:

Conversion of integer to a char variable

```
int a = 20;
char ch = (char) a;
```

Note: The above conversion is valid as after all characters are integer values of ASCII codes.

innovate achieve lead

BITS Pilani
Pilani Campus

Operators and Expressions

Celsius to Fahrenheit Program

```
#include <stdio.h>
int main()
{
  float cel, far;         /* variable declarations */

  printf("Enter the temperature in deg. Celsius: ");

  scanf("%f", &cel);    /* getting user input */

  far = cel * 1.8 + 32;

  printf("%f degree C = %f degree F\n\n", cel, far);
                   /* printing the output */
  return 0;

}
```

Operators

- Can be unary, binary or ternary
- Used to perform some operation (e.g., arithmetic, logical, bitwise, assignment) on operands
- In an expression with multiple operators, the order of evaluation of operators is based on the **precedence** level
- Operators with the same precedence work by rules of **associativity**
- C does not define the order in which the operands of an operator will be evaluated

<table>
<tr><th>Operator</th><th>Description</th><th>Associativity</th></tr>
<tr><td>()
[]
.
->
++ --</td><td>Parentheses (grouping)
Brackets (array subscript)
Member selection via object name
Member selection via pointer
Unary post-increment/post-decrement</td><td>left-to-right</td></tr>
<tr><td>++ --
+ -
! ~
(type)
*
&
sizeof</td><td>Unary pre-increment/pre-decrement
Unary plus/minus
Unary logical negation/bitwise complement
Unary cast (change type)
Dereference
Address
Determine size in bytes</td><td>right-to-left</td></tr>
<tr><td>* / %</td><td>Multiplication/division/modulus</td><td>left-to-right</td></tr>
<tr><td>+ -</td><td>Addition/subtraction</td><td>left-to-right</td></tr>
<tr><td><< >></td><td>Bitwise shift left, Bitwise shift right</td><td>left-to-right</td></tr>
<tr><td>< <=
> >=</td><td>Relational less than/less than or equal to
Relational greater than/greater than or equal to</td><td>left-to-right</td></tr>
<tr><td>== !=</td><td>Relational is equal to/is not equal to</td><td>left-to-right</td></tr>
<tr><td>&</td><td>Bitwise AND</td><td>left-to-right</td></tr>
<tr><td>^</td><td>Bitwise exclusive OR</td><td>left-to-right</td></tr>
<tr><td>|</td><td>Bitwise inclusive OR</td><td>left-to-right</td></tr>
<tr><td>&&</td><td>Logical AND</td><td>left-to-right</td></tr>
<tr><td>||</td><td>Logical OR</td><td>left-to-right</td></tr>
<tr><td>?:</td><td>Ternary conditional</td><td>right-to-left</td></tr>
<tr><td>=
+= -=
*= /=
%= &=
^= |=
<<= >>=</td><td>Assignment
Addition/subtraction assignment
Multiplication/division assignment
Modulus/bitwise AND assignment
Bitwise exclusive/inclusive OR assignment
Bitwise shift left/right assignment</td><td>right-to-left</td></tr>
<tr><td>,</td><td>Comma (separate expressions)</td><td>left-to-right</td></tr>
</table>

Types of Operators

- Arithmetic operators
- Relational operators
- Logical operators
- Bitwise operators
- Assignment operators
- Conditional operators
- Special operators

Arithmetic Operators

- Used for performing arithmetic operations
- Binary arithmetic operators
  - Takes two operands as input
  - *, /, %, + and −
  - *, /, % have higher precedence than + and −

Arithmetic Operators

- int a = 31, b = 10;
- floats c = 31.0, d = 10.0;

For integers:

a + b = 41
a – b = 21
a/b = 3
a%b = 1
a * b = 310

For float:

c + d = 41.000000
a – b = 21.000000
c/d = 3.100000
c%d = Not valid
c * d = 310.000000

Arithmetic Operators

- Unary arithmetic operator
  - Performs operation on single operand
  - ++ (Increment operator) and -- (Decrement operator)
  - Both these operator can be applied before an operand as well as after the operand
  - All arithmetical operators follow left to right associativity

Arithmetic Operators

Increment (++) and Decrement (--) operators

- Can be used either as prefix or postfix
  - Prefix:
    - "Change value and then use"
    - Lower precedence than the postfix
  - Postfix:
    - "Use and then change value"
    - Higher precedence than prefix
- Can be applied only to variables
- Causes side effects

Increment and Decrement Operators

Prefix:

```
++<variable_name> / --<variable_name>
```

Example:

```
int  A = 10;
printf("A is %d ", ++A);
int B = --A;
printf("B is %d ", B);
```

First, the value is increased/decreased and then used

Output: A is 11 B is 10

Increment and Decrement Operators

Postfix:

```
<variable_name>++ / <variable_name>--
```

Example:

```
int  A = 10;
printf("A is %d", A++);
int B = A--;
printf("B is %d", B);
```

First, the value is used and then increased (or decreased)

Output: A is 10 B is 11

Relational Operators

- `a == b` → checks whether a is equals to b
- `a != b` → checks whether a is not equal to b
- `a < b` → checks whether a is less than b
- `a > b` → checks whether a is greater than b
- `a <= b` → checks whether a is less than equal to b
- `a >= b` → checks whether a is greater than equal to b

- All are of same precedence and are left to right associative

Logical Operators

- `expr`$_1$ `&&` `expr`$_2$ → Returns true, when both operands are non-zero

- `expr`$_1$ `||` `expr`$_2$ → Returns true, when at least one of the expression is non-zero

- `!(expr)` → If `(expr)` is non zero, then `!(expr)` returns zero

- All are of same precedence and are left to right associative

Example

```
int a = 10, b = 4, c = 10, d = 20;
```

```
Evaluate the expression:(a > b && c == d)
```

```
Evaluate the expression:(a > b || c == d)
```

```
Evaluate the expression: (!a)
```

Bitwise Operators

- Performs operation at bit level

- `a | b` → Known as bitwise OR. Sets the bit when the bits in at least one of the operand is set

- A = 43 and B = 7

- A | B = 0 0 1 0 1 0 1 1 (43)

  0 0 0 0 0 1 1 1 (7)

  = 0 0 1 0 1 1 1 1 (47)

Bitwise Operators

- `a & b` → Known as bitwise AND. Sets the bit only when the bits in both the operands are set

- A = 43 and B = 7

- A & B = 0 0 1 0 1 0 1 1 (43)

  0 0 0 0 0 1 1 1 (7)

  = 0 0 0 0 0 0 1 1 (3)

Bitwise Operators

- `a^b` → Known as bitwise XOR. Sets the bit only when the bit is set in only one of the operand

- A = 43 and B = 7

- A ^ B = 0 0 1 0 1 0 1 1 (43)
  0 0 0 0 0 1 1 1 (7)
  = 0 0 1 0 1 1 0 0 (44)

Bitwise Operators

- `~a` → Flip the bits. Sets the bit only when the bit in the operand is not set.

- A = 43 = 0 0 1 0 1 0 1 1 (43)

  ~A = 1 1 0 1 0 1 0 0 (-84)

- A = -63 = 1 1 0 0 0 0 0 1 (-63)

  ~A = 0 0 1 1 1 1 1 0 (62)

Assignment Operator

- `A = B // assigns value of B to the variable A`
- `A op= B` → `A = A op B`, `op` can be any binary arithmetic or bitwise operator

E.g., A = 43, B = 7

A += B → A = A + B O/P: A = 50, B = 7

A &= B → A = A & B O/P: A = 3, B = 7

Important Considerations

- C does not specify the order in which the operands of an operator are evaluated.
- Exceptions: `&&` `||` `?:` `,`
- The operands of **logical-AND** and **logical-OR** expressions are **evaluated from left to right**. If the value of the first operand is sufficient to determine the result of the operation, the second operand is not evaluated. This is called "*short-circuit evaluation*".
- If conditional operator also has order of evaluation:
  - Example: `int i=(a<10)?10:20;`

Short Circuiting in C

- A short circuit in logic is when it is known for sure that an entire complex conditional is either true or false before the evaluation of whole expression

- Mainly seen in case of expressions having logical AND(&&) or OR(||)

Short Circuiting in Logical AND

Consider the expression: `E1 && E2`

- Output of the expression is true only if both `E1` and `E2` are non-zero
- If `E1` is false, `E2` never gets evaluated

```
a = 0, b = 3;
int I = ++a && ++b;
printf("%d %d %d", a, b, I);
```

O/P ➔ 1, 4, 1

```
a = 0, b = 3;
int I = a++ && ++b;
printf("%d %d %d", a, b, I);
```

O/P ➔ 1, 3, 0

Short Circuiting in Logical OR

Consider the expression: `E1 || E2`

Output of the expression is false if only both `E1` and `E2` are zero

If `E1` is true, there is no need to evaluate `E2`

```
a = 0, b = 3;
int I = ++b || ++a;
printf("%d %d %d", a, b, I);
```

O/P → 0, 4, 1

```
a = 0, b = 3;
int I = ++a || ++b;
printf("%d %d %d", a, b, I);
```

O/P → 1, 3, 1

Problems to Solve

```
int a = 1, b = 1;
int c = a || --b; // --b is not evaluated
int d = a-- && --b; //  --b is evaluated,  b becomes 0
printf("a = %d, b = %d, c = %d, d = %d", a, b, c, d);
```

O/p : a = 0, b = 0, c = 1, d = 0

```
int i=-1,j=-1,k=0,l=2,m;
m = i++ && j++ && k++ || l++;
printf("i = %d, j = %d, k = %d, l = %d, m =
   %d",i,j,k,l,m);
```

O/p : i = 0, j = 0, k = 1, l = 3, m = 1

innovate achieve lead

BITS Pilani
Pilani Campus

Thank you

Q & A