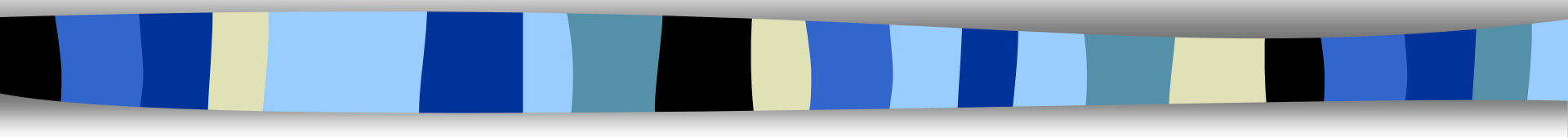
Introduction to C Programming



Introduction

History of C

- Evolved from two previous languages
  - BCPL , B
- BCPL (Basic Combined Programming Language) used for writing OS & compilers
- B used for creating early versions of UNIX OS
- Both were "***typeless***" languages
- C language evolved from B (Dennis Ritchie – Bell labs)

** **Typeless – no datatypes. Every data item occupied 1 word in memory**.

History of C

- Hardware independent
- Programs portable to most computers
- **Dialects of C**
  - Common C
  - ANSI C
    - ANSI/ ISO 9899: 1990
    - Called American National Standards Institute ANSI C
- Case-sensitive

C Standard Library

- Two parts to learning the “C” world
  - Learn C itself
  - Take advantage of rich collection of existing functions called C Standard Library
- Avoid reinventing the wheel
- SW reusability

Basics of C Environment

- C systems consist of 3 parts
  - Environment
  - Language
  - C Standard Library
- Development environment has 6 phases
  - Edit
  - Pre-processor
  - Compile
  - Link
  - Load
  - Execute

Simple C Program

```
/* A first C Program*/

#include <stdio.h>

void main()

{
        printf("Hello World \n");

}
```

Simple C Program

- **Line 1: #include <stdio.h>**

- As part of compilation, the C compiler runs a program called the **C preprocessor**. The preprocessor is able to add and remove code from your source file.
- In this case, the **directive #include** tells the preprocessor to include code from the file **stdio.h.**
- This file contains declarations for functions that the program needs to use. A declaration for the **printf** function is in this file.

Simple C Program

- **Line 2: void main()**

- This statement declares the **main function**.
- A C program can contain many functions but must always have one main function.
- A function is a self-contained module of code that can accomplish some task.
- Functions are examined later.
- The "void" specifies the return type of main. In this case, nothing is returned to the operating system.

Simple C Program

- **Line 3: {**

- This opening bracket denotes the start of the program.

Simple C Program

- **Line 4: printf("Hello World From About\n");**

- **Printf** is a function from a standard C library that is used to print strings to the standard output, normally your screen.
- The compiler links code from these standard libraries to the code you have written to produce the final executable.
- The **"\n"** is a special format modifier that tells the **printf** to put a line feed at the end of the line.
- If there were another **printf** in this program, its string would print on the next line.

Simple C Program

- **Line 5: }**
- This closing bracket denotes the end of the program.

Escape Sequence

- \n new line
- \t tab
- \r carriage return
- \a alert
- \\ backslash
- \” double quote

Memory concepts

- Every variable has a name, type and value
- Variable names correspond to locations in computer memory
- New value over-writes the previous value– “Destructive ***read-in***”
- Value reading called “Non-destructive ***read-out***”

Arithmetic in C

C operation	**Algebraic**	**C**
Addition(+)	f+7	f+7
Subtraction (-)	p-c	p-c
Multiplication(*)	bm	b*m
Division(/)	x/y, x , x y	x/y
Modulus(%)	r mod s	r%s

Precedence order

- **Highest to lowest**
  - ()
  - *, /, %
  - +, -

Example

Algebra:

$$z = pr\%q+w/x-y$$

C:

```
z = p * r % q + w / x – y ;
```

Precedence:

1 2 4 3 5

Example

Algebra:

a(b+c)+ c(d+e)

C:

a * (b + c) + c * (d + e) ;

Precedence:

3 1 5 4 2

Decision Making

- Checking falsity or truth of a statement
- Equality operators have lower precedence than relational operators
- Relational operators have same precedence
- Both associate from left to right

Decision Making

- Equality operators
  - ==
  - !=
- Relational operators
  - <
  - >
  - <=
  - >=

Operator	Description	Precedence level	Associativity
( ) [ ] → .	Function call Array subscript Arrow operator Dot operator	1	Left to Right
+ - ++ - - ! ~ * & (datatype) sizeof	Unary plus Unary minus Increment Decrement Logical NOT One's complement Indirection Address Type cast Size in bytes	2	Right to Left
* / %	Multiplication Division Modulus	3	Left to Right
+ -	Addition Subtraction	4	Left to Right
<< >>	Left shift Right shift	5	Left to Right
< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	6	Left to Right
= = !=	Equal to Not equal to	7	Left to Right
&	Bitwise AND	8	Left to Right
^	Bitwise XOR	9	Left to Right
|	Bitwise OR	10	Left to Right
&&	Logical AND	11	Left to Right
||	Logical OR	12	Left to Right
?:	Conditional operator	13	Right to Left
= *= /= %= += -= &= ^= |= <<= >>=	Assignment operators	14	Right to Left
,	Comma operator	15	Left to Right

Assignment operators

- =
- +=
- -=
- *=
- /=
- %=

Increment/ decrement operators

- ++ ++a
- ++ a++
- -- --a
- -- a--

Increment/ decrement operators

```
main()
{
        int c;
        c = 5;
        printf("%d\n", c);
        printf("%d\n", c++);
        printf("%d\n\n", c);

        c = 5;
        printf("%d\n", c);
        printf("%d\n", ++c);
        printf("%d\n", c);

  return 0;
}
```

```
5
5
6

5
6
6
```

Thank You

- Thank You