# Programming Languages

Asst. Prof. Dr. Irem Turkmen

## COMPUTER LANGUAGES

- Machine Languages /* FIRST */
  - 000000 00001 00010 00110 00000 100000
- Assembly Languages /* 1940s and 1950s */
  - LOAD B, %r0
  - LOAD C, %r1
  - ADD %r0, %r1
  - SUB &2, %r1
  - STORE %r1, a
- High-Level Languages /* 1970s – C,Java,.. */
  - a=b+c-2;

# HIGH-LEVEL LANGUAGES

- Translating High-Level Programming Code to Machine Language
  - Compiler
  - Interpreter
- Advantages of High-Level Languages
  - Portability
  - Readability
  - Maintainability -> Easy to modify and debug

# History of C

- 1972
- AT&T Bell Labs
- Dennis M. Ritchie
- Operating System, Systems Programming Language
- UNIX, written with B (C's predecessor)

## What is C used for?

- System programming
- Operating Systems like Linux
- Microcontrollers: automobiles and airplanes
- Embedded processors : phones, portable electronics, etc.
- DSP processors: digital audio and TV systems
- …

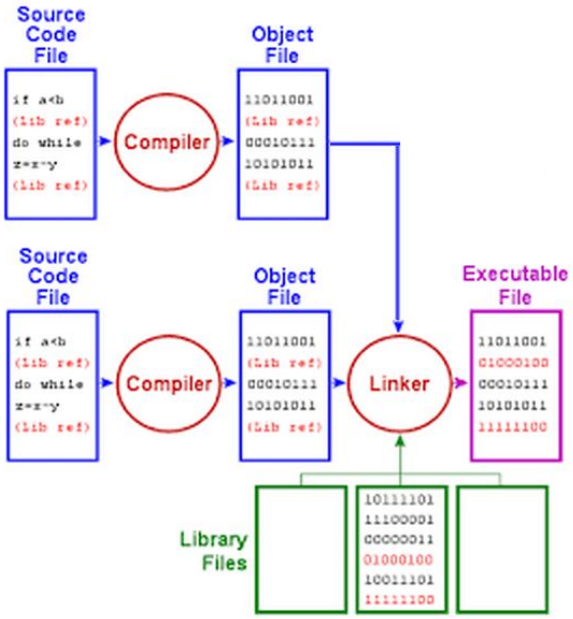
## Features of C

- Few keywords
- Structures, unions – compound data types
- Pointers – memory, arrays
- External standard library – I/O, other facilities
- Macro preprocessor

## Nature Of C

- Powerful but dangerous
- No range checking
- No type checking at runtime
- No garbage collector

## Compilation Process



# Compilers / Editors&IDEs

- IDE (Integrated Development Environments)
  - DevC++ (gcc compiler), http://www.bloodshed.net
    - Windows, Linux
  - Code::Blocks (gcc, Borland CC, Digital Mars, Open Watcom), http://www.codeblocks.org
    - Windows, Linux, Mac OS X
  - Xcode (gcc), https://developer.apple.com/xcode/
    - Mac OS X
  - Eclipse, http://www.eclipse.org/cdt/
- Editors
  - Ultrapad
  - Notepad++

# Writing C Programs in Linux

- .c extension
- gcc compiler
- Terminal
  - gcc main.c –o mainP
  - ./mainP

## First C Program

```
/* An example C source code*/                    // Comments

# include<stdio.h>                               //Preprocessor Command

  int main() {                                   //Main Function
        int num1, num2;                          //Variable Declarations
        int sum;
        scanf ("%d %d",&num1,&num2)
        sum=num1+num2;                           //Expressions & statements
        printf("the sum=%d",sum);
        return 0;
  }
```

## comments

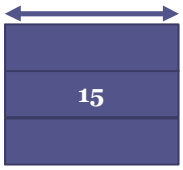
- A comment is text that you include in a source file to explain what the code is doing!
  - Comments are for human readers – compiler ignores them!
- The C language allows you to enter comments between the symbols /* and */
- // can be used for a single line comment
- What to comment ?
  - Function header
  - Changes in the code

```
/* square()
   Author : P. Margolis
   Initial coding : 3/87
   Params : an integer
*/

 //Returns : square of its parameter
```

## variables & literals

- The statement
  - j = 15+10;
- ***A literal*** is written number that never changes
- ***A variable*** achieves its variableness by representing ***an address***, in computer memory.



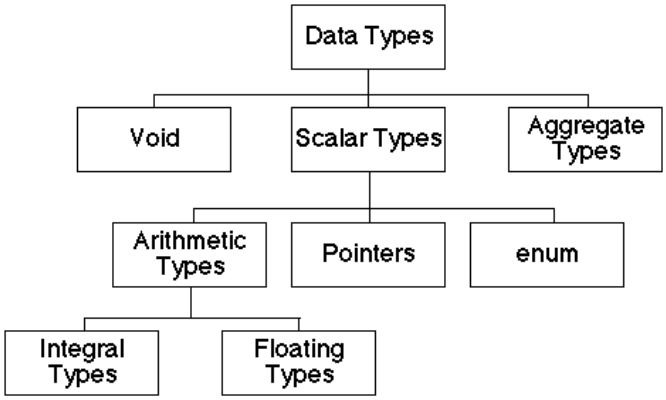

## names

- In the C language, you can name just about anything: variables, constants, functions, and even location in a program.
- Names may contain letters, numbers, and the underscore character ( _ )
  - but must start with a letter or underscore…
- The C language is case sensitive which means that it differentiates between lowercase and uppercase letters
  - VaR, var, VAR
- A name can NOT be the same as one of the ***reserved keywords***.

## Reserved Keywords

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

## Data Types



## data types and declaration

- Basic arithmetic types
  - char, int, float, double
- Qualifiers
  - long, short, signed(default), unsigned
- You can declare variables that have the same type in a single declaration
  - int j,k; (j and k are signed ints by default)
  - ***All declarations in a block must appear before any executable statements***

```
int j;
unsigned char a;
short int num;
```

## different types of integrals

- A byte must be ***at least 8 bits long***,
- Ints must be ***at least 16 bits long*** and must represent the "***natural***" size for computer.
  - natural means the number of bits that the CPU usually handles in a single instruction

| Type | Size (in bytes) | Value Range |
|---|---|---|
| int | 4 | $-2^{31}$ to $2^{31}-1$ |
| short int | 2 | $-2^{15}$ to $2^{15}-1$ |
| long int | 4 | $-2^{31}$ to $2^{31}-1$ |
| unsigned short int | 2 | 0 to $2^{16}-1$ |
| unsigned long int | 4 | 0 to $2^{32}-1$ |
| signed char | 1 | $-2^{7}$ to $2^{7}-1$ |
| unsigned char | 1 | 0 to $2^{8}-1$ |

# floating point types

- to declare a variable capable of holding floating-point values
  - **float**
  - **double**
- The word ***double*** stands for double-precision
  - A float generally requires ***4 bytes***, and a double generally requires ***8 bytes***

    float: E-38 E+38

    double: E-308 E+308
  - ***read more about limits in <limits.h>***
- Decimal point
  - 0.356
  - 5.0
  - 0.000001
  - .7
  - 7.
- Scientific notation
  - 3e2
  - 5E-5

# initialization

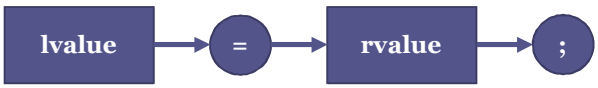
- A declaration allocates memory for a variable, but it does not necessarily store an initial value at the location
- To initialize a variable, just include an assignment expression after the variable name
  - char c = 'A' ;
- It is same as
  - char c;
  - c = 'A';
- Constants: fixed values that are used in a program
  - const int pi=3;

## Expressions

- An ***expression*** is any combination of operators, numbers, and names that donates the computation of a value.
- **Examples**
  - 5 A literal
  - j A variable
  - 5 + j literal plus a variable
  - f() A function call
  - f()/4 A function call, whose result is divided by a literal

## ASSIGNMENT STATEMENTS

- The left hand side of an assignment statement, called an ***lvalue***, must evaluate to a memory address that can hold a value.
- The expression on the right-hand side of the assignment operator is sometimes called an ***rvalue***.




answer = num * num;

number = i+5;

num * num = answer

i+1 =number+2;

# PRINTF() and SCANF() functions

- The printf() function can take any number of arguments.
  - The first argument is called as ***format string***. It is enclosed in double quotes and **may contain** text and ***format specifiers***
  - ***Format specifiers*** define the type of data to be printed or scaned
    ***%d -> int (decimal)***
    ***%f -> floating point***
    ***%c -> characters***
    ***%s -> strings***

```
int num;
num=5;
printf(“number : %d”, num);
```

```
number: 5
```

```
float n=3.14
char m=‘a’;
printf("the value of n : %f m:%c",n,m);
```

```
the value of n:3.14 m:a
```



# PRINTF() and SCANF() functions

- The scanf() function is the mirror image of printf(). Instead of printing data on the terminal, it reads data entered from keyboard.
  - The first argument is a format string that contain format specifier.
  - **The major difference between scanf() and printf() is that the data item arguments must be lvalues in scanf()**

```
int num;
scanf(“%d”, &num);
printf(“num : %d\n”, num);
```

```
float n,m;
scanf("%f %f",&n,&m);
printf("n:%f m:%f",n,m);
```

# Sample C Code

```
// this program calculates the area of a circle
#include <stdio.h>
int main()
{
        const float pi=3.1415;
        int c;
        float area;
        printf("please give the radius\n");
        scanf("%d",&c);
        area=c*c*pi;
        printf("the area of the circle with a
radius %d is %f",c,area);
        return 0;
}
```