**C Language Notes**

Date: 4/Aug/2022 Lecture 2

**Introduction**

I mid 60s BCPL (Basic Combined Programming Language) was formed by Martin Richards in 1966 at AT&T Laboratories. Some researchers from AT&T lab, GE lab and Massachusetts University decided to build an operating system named as MULTICS but later AT&T lab disagreed with the coalition’s model of Operating System.

So AT&T lab decided to build an operating system by self so they made a team led by Ken Thompson. Ken Thompson thought BCPL language would be better to build an operating system but they found BCPL must be upgraded so they made B language (letter B is decided to honor Mr. Martin). Later **Dennis Ritchie** improved B Language and found a new language as C Language.

**Version History**

* K & R: by Dennis Cunningham and Ritchie in **1978.**

From 1972 to 1978 “*X=-5* ***means*** *X=X-5”*, While *X= -5(space b/w ‘=’ &’ -‘) represents X assigns ‘-5’.*

*After X-=5 meant X=X-5.*

* ANSIC: 1983
* C11- 2011 ~~gets()~~ fgets().
* C17- 2018.
* C has evolved over the years with new versions and updates. Here is a brief overview of the evolution of C version by version:
* C Version 0: This was the first version of C developed by Dennis Ritchie in 1972 for use on the DEC PDP-11 computer. It was a very simple language that lacked many of the features that later versions of C would come to include.
* C Version 1: This version of C was released in 1978 and included a number of new features, such as support for signed and unsigned integers, type qualifiers, and the void keyword.
* C Version 2: This version of C was released in 1983 and included a number of significant new features, such as support for function prototypes, enumerated types, and the const keyword.
* C Version 3: This version of C was released in 1989 and included a number of minor updates and bug fixes. It was not widely adopted, however, as most developers were using ANSI C at this point.
* ANSI C: ANSI C, also known as C89, was a standardized version of C released in 1989 by the American National Standards Institute (ANSI). It included a number of new features, such as support for function prototypes, new standard library functions, and stricter rules for type checking and type promotion.
* C90: C90, a lso known as ISO/IEC 9899:1990, was an international standard for C released in 1990. It was based on ANSI C and included a number of minor updates and clarifications.
* C99: C99, also known as ISO/IEC 9899:1999, was a major update to the C language released in 1999. It included a number of significant new features, such as support for variable-length arrays, flexible array members, inline functions, and designated initializers.

*Ex: Single line command, variable length array’s. 2. a[n] can also be used as declaration in the place of a[5] or something.*

* C11: C11, also known as ISO/IEC 9899:2011, was released in 2011 and included a number of new features, such as support for thread-local storage, anonymous structures and unions, improved support for multi-threading, and additional math functions in the standard library.
* C17: C17, also known as ISO/IEC 9899:2018, was released in 2018 and included a number of minor updates and bug fixes.
* In summary, C has evolved over the years with each new version adding new features, clarifications, and bug fixes. While newer versions of C have introduced significant new features, the core syntax and semantics of the language have remained largely unchanged, making it a stable and reliable choice for system programming and application development.

**Begin C**

**Tokens**

All the symbols used in C language are known as Tokens. i.e., A to Z, a to z, 0 to 9, {,[,?,~,<,>,+,/,\,(,) etc.

Types of Tokens

* Constants

Data = Information = Constant

Types of Data/Constants

* + - 1. Primary Constant
         1. Integer: 25,-5, 42, -236.
         2. Real: 3.7, 3.0.
         3. Character: ‘a’, ‘A’, ‘+’ (*single quotes is mandatory to symbolize a character constant.)* Length of character const. must be 1.
      2. Secondary Constants
         1. Arrays
         2. String
         3. Pointer
         4. Structure
         5. Union
         6. Enumerator
* Variables

Name of a memory location to be reserved for any specific process

a b c

Program’s Memory

MU

CU

ALU

Processor RAM

**a, b, c are variables here.**

**Use of CU, MU and ALU**

Statements in MU sends to CU to be read then CU signals to ALU to do calculations then in return ALU sends the calculated or processed data to CU then it assigns the data in MU.

**Variable Naming Rules**

* + - 1. Variable name is any combination of alphabet, digit and underscore. i.e., a25, A25 (case sensitive).
      2. No Other symbol is allowed.
      3. Variable name cannot start with a digit. i.e., ~~25a, 25A~~.
* Keywords
  + - 1. C89

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

* + - 1. C99 (+5) : **Bool, Complex, Imaginary, inline, restrict.**
      2. C11 (+7): \_**Aligans, \_Alignof, \_Atomic, \_Generic, \_Noretum, \_Static\_asset, \_Thread\_local.**
* Operator
* Functions

Functions are block of statements, which has some name for identification. Function name must be unique.

**Instruction**

Instructions are rules or method to make the meaningful Statements known as syntax.

* Declaration Statements.
* Action Statements.

5/Aug/2022 Lecture 3

**Data Types**

* Size required storing data.
* Internal binary representation of data.
* Kind of operation.

1. **Primitive Data Types:** The type of data type which are already build in C language i.e., int, char, float, double.
2. **Non-Primitive:** The type of data types which are built manually by the programmer.

**Variable Declarations**

We have to declare variables as required.

Example: 1. int a,b; ( Each consume 4 byte afterwards C89 but before C89 int consumed 2 variables). “int” data type stores Integer constant.

2. char m; (Each consume 1 byte). “char” data type stores Character constant.

3. float k; (Each consume 4 byte).”float” data type stores real constant.

4. double d1; (Each consume 8 bytes). “double” data type also stores real constant.

**Block Structure**

* C is a block structured programming language.
* A block is a group of instructions.
* Outer blocks are usually functions.
* Functions are block of statements, which has some name for identification.
* A C program can have any number of blocks.
* Even in the smallest C program, there is at least one function.
* If there is only one function in the program then its name must be main().
* You can write declaration statements outside the function body, but action statements must be written inside the function body.

**Action Statements**

1. **Output Instruction**

Such instruction that used to take output when executed.

**Printf():** This predefined function is used to print a text on the monitor. It can be used to print value of variable or an expression.

**Example:** #include<stdio.h>

void main(){

printf(“Hello Students”);

}

**Escape Sequences**

\n New line

\t Tab Space

\b Backspace

\r Carriage Return

\\ Print \

\” Print “

\’ Print ‘

**Format Specifier**

%d int

%c char

%f float

%lf double

1. **Input Instruction:**

Such instructions which are used to fetch data from the input device.

**getch():** It can take only one character at a time.

**scanf():**

* It can take data which requires multiple key strokes.
* It can take data as a sequence of characters and uses space, tab and new line character as data separator.

**Delimiters:** Space, tap and enter.

* It can convert data into desired type.
* It can store data in specified variable.

Example: #include<stdio.h>

#include<conio.h>

int main(){

int a;

scanf(“%d”,&a);

printf(“a=%d”,a);

getch();

}

Note: Every time scanf() take any data it takes as character sequence.

1. **Arithmetic Instruction**

An instruction which is used to manipulate data using operators is known as arithmetic instruction.

*Various Operators with their priority sequence*

* 1. Unary Operators

+, -, ++, --, sizeof() etc.

* + 1. ++ (Increment Operator): It increases the value of a variable by one unit.
       1. Pre Increment: int a;

++a;

(Most prior unary operator.)

#include<stdio.h>

#include<conio.h>

int main(){

int a=5,b;

b=++a;

printf("%d %d",a,b);

getch();

}

***VALUE: 6 6***

* + - 1. Post Increment: int a;

a++;

(Least prior unary operators.)

#include<stdio.h>

#include<conio.h>

int main(){

int a=5,b;

b=a++;

printf("%d %d",a,b);

getch();

}

***VALUE: 5 6***

* + 1. -- (Decrement Operator): It decreases the value of a variable by one unit.
       1. Pre Decrement: int a;

++a;

(Most prior unary operator.)

#include<stdio.h>

#include<conio.h>

int main(){

int a=5,b;

b=--a;

printf("%d %d",a,b);

getch();

}

***VALUE: 4 4***

* + - 1. Post Decrement: int a;

a++;

(Least prior unary operators.)

#include<stdio.h>

#include<conio.h>

int main(){

int a=5,b;

b=a--;

printf("%d %d",a,b);

getch();

}

***VALUE: 4 5***

* + 1. sizeof() operator: It gives the size of any variable in byte.
  1. Arithmetic Operators

\*, /, % are equal prior and operation goes left to right but more prior than + and -.

* 1. Bitwise Operators
     1. & (And Operator)
        1. 0 & 0: 0
        2. 0 & 1: 0
        3. 1 & 0: 0
        4. 1 & 1: 1
     2. | (Or Operator)
        1. 0 & 0: 0
        2. 0 | 1: 1
        3. 1 | 0: 1
        4. 1 | 1: 1
     3. ^ (XOR Operator)
        1. 0 ^ 0: 0
        2. 0 ^ 1: 1
        3. 1 ^ 0: 1
        4. 1 ^ 1: 0
     4. ~ (NOT Operator)
        1. ~0: 1
        2. ~1: 0
     5. >> (Right Shift Operator)
     6. >> (Left Shift Operator)
  2. Relational Operators

<, >, <=, >=, ==, != are relational operators.

* 1. Logical Operators
     1. ! NOT (UNARY)
     2. && AND
     3. || OR
  2. Conditional Operators
  3. Assignment Operators

=, +=, \*=, /=, %= are some assignment operators.

1. **Control Instruction**
   1. Decision Control Instruction (Selection control Instruction)

Program builds once but executes many time. So to make program capable of taking decisions we use decision control Instruction.

**Various Control Instructions**

* + 1. if

Explanation: *if(condition)*

*{*

*code;*

*code;*

*code;*

*}*

If condition is true then code in ‘if block’ will run, otherwise if false condition is false then no code from if block will be run.

Rule: if condition is giving any non-zero value then it will considered as true and zero value is considered as false.

Example: *#include<stdio.h>*

*#include<conio.h>*

*int main(){*

*int x;*

*printf("Enter any number: ");*

*scanf("%d",&x);*

*if (x>0){*

*printf("Number is +ve");*

*}*

*if(x==0){*

*printf(“Neither +ve or -ve”);*

*}*

*if (x<0){*

*printf("Number is -ve");*

*}*

*getch();*

*}*

* + 1. if else

Explanation: if(condtion){

code;

code;

code;

}

else{

code;

code;

code;

}

If condition is true then code in ‘if block’ will run, otherwise if condition is false then else block will run.

Example: *#include<stdio.h>*

*#include<conio.h>*

*int main(){*

*int x;*

*printf("Enter any number: ");*

*scanf("%d",&x);*

*if (x>0){*

*printf("Number is +ve");*

*}*

*else{*

*printf("Number is non +ve");*

*}*

*getch();*

*}*

* + 1. ?: (Conditional Operator)

Explanation: It is an *Ternary Operator*.

*exp1? exp2:exp3*

Example: *#include<stdio.h>*

*#include<conio.h>*

*int main(){*

*float x;*

*printf("Enter any number: ");*

*scanf("%f",&x);*

*/\*x>0?printf("Number is +ve"):printf("Number is non +ve");*

*OR\*/*

*printf(x>0?"Number is +ve":"Number is non +ve");*

*getch();*

*}*

* Nested if/Nested if-else

|  |  |
| --- | --- |
| Nested If  *if(condition){*  *if(condition){*  *…………………*  *}*  *}* | Nested If-else  *if(condition){*  *if(condition){*  *…………………*  *}*  *else{*  *…………………*  *}*  *}*  *else{*  *if(condition){*  *…………………*  *}*  *else{*  *…………………*  *}*  *}* |
| Nested If  *if(condition){*  *if(condition){*  *…………………*  *}*  *else{*  *………………..*  *}*  *}* |

* if else ladder

|  |  |
| --- | --- |
| if(condition)  ………………  else{  if(condition)  …………….  else{  if(condition)  ……………..  else  ……………  }  } | if(condition)  ………………..  else if(condition)  ………………..  else if(condition)  ………………..  else  ………………. |

* 1. Iterative Control Instruction (Loop).
     1. while(entry control loop)

*while( condition ){*

*……………;*

*……………;*

*}*

If condition is found to be true then the code in while block will be execute, then again check condition, if true, then code will be executed again and again unless condition will false.

* + 1. do while(exit control loop)

do{

……………..;

……………..;

……………..;

}while(condition);

*/\* int i=1;*

*do{*

*printf(“%d”,i);*

*i++;*

*}while(i<=10);*

First the code will run of “do” block and then condition in while be checked if condition is true then then do block will run else it will be terminate and move to next block.

* In this loop the code will be executed at least once.
  + 1. for (entry control loop)

for(intialisation;condition;flow)

{

…………..;

…………..;

}

/\**for(i=1;i<=10;i++){*

*printf(“%d”,i);*

*}*

***“break”***

* *“break”* is a keyword.
* “break” can be used either in loop’s body or switch’s body.
* In loop break is used to terminate execution of loops.

*int i=1;*

*while(age<=100){*

*……………..;*

*……………..;*

*If(condition)*

*break;*

*……………..;*

*……………..;*

*age++;*

*}*

* 1. Switch case control Instruction.

Syntax:

*switch(expression){*

*case constant:*

*……………..;*

*case constant:*

*……………..;*

*case constant*

*..…………..;*

*.*

*.*

*default:*

*default code.;*

*}*

\*Note: 1.expression’s output must be integer.

2. The code of case will run whose constant will be the same as output integer.

3. The code will run unless the all the cases of the switch control instruction. To stop the execution you will have to use the ‘*break*’keyword to stop the execution.

* 1. Goto Control Instruction.

**Function**

Function is a block of code, which has some name for identification.

**Synatax:** *function\_name(){*

**……………………..; function definition.**

**..…………………...;**

*}*

* Even in the smallest C program, there is atleast one function.
* All function names must be unique.
* One function name must be main().
* You can define functions in any sequence.
* No keyword is a function.

Function are of two types:

* Predefined Functions: The functions whose machine code(already compiled) is already stored in the C libraries.
* User defined Function:

User defined functions has three stages:

Declaration: First we have to declare a function to be called.

Definition: We have to define its statements to execute when it is called.

Call: To execute the statements of the function we have to call it.

* + Variables declared in the function are known as **local variables**.

**There are four types of user defined functions:**

* + 1. **Takes Nothing, Returns Nothing(TNRN):**

**Example:** *void add();* **Function declaration, Empty parenthesis means TN.**

*add(){*

*int a,b;*

*printf(“Enter two numbers”);* **Function Definition, No**

*scanf(“%d%d”,&a,&b);* **return keyword means RN.**

*printf(“Sum is %d”,a+b);*

*}*

*int main(){*

*add();* **Function Call**

*return 0;*

*}*

* + 1. **Takes Something, Returns Nothing(TSRN):**

**Example:** *void add(int,int);* **Function declaration, TS**

*void add(int a, int b){* **Function Definition, No return**

*printf(“Sum is %d”,a+b);* **keyword means RN.**

*}*

*int main(){*

*int a,b;*

*printf(“Enter two numbers”);*

*scanf(“%d%d”,&a,&b);*

*add(a,b);* **Function Call**

*return 0;*

*}*

* + 1. **Takes Nothing, Returns Something(TNRS):**

**Example:** *int add();* **Function declaration, Empty parenthesis means TN.**

*int**add(){*

*int a,b;*

*printf(“Enter two numbers”);* **Function Definition,**

*scanf(“%d%d”,&a,&b);* **return keyword means RS.**

*return (a+b);*

*}*

*int main(){*

*int c;*

*c=add();* **Function Call**

*printf(“Sum: %d”,c);*

*return 0;*

*}*

* + **NOTE: We cannot return more than one value using return keyword.**
    1. **Takes Something Return Something(TSRS):**

**Example:** *int add(int,int);* **Function declaration, TS**

*void add(int a, int b){* **Function Definition, Return**

*return( a+b);* **keyword means RS.**

*}*

*int main(){*

*int a,b,c;*

*printf(“Enter two numbers”);*

*scanf(“%d%d”,&a,&b);*

*c=add(a,b);* **Function Call**

*printf(“Sum: %d”,c);*

*return 0;*

*}*

**Recursion**

**What is recursion?**

* Function calling itself is called recursion.
* A recursive method solves a problem by calling a copy of it to work on a smaller problem.
* It is important to ensure that the recursion terminates.
* Each time the function call itself with a slightly simpler version of the original problem.

**Recursive Tree:**

3

K

main()

**k=f1(3);**

**printf(“%d”,k);**

*int main(){*

int f1(int n)

*int k;*

**int(n==1)**

**return 1;**

**s=n+ f1(n-1);**

**return (s);**

*k=f1(3);*

3

*printf(“%d”,k);*

n

*}*

*int f1(int n){*

2

s

*int s;*

*if(n==1)*

int f1(int n)

**int(n==1)**

**return 1;**

**s=n+ f1(n-1);**

**return (s);**

*return 1;*

s

n

2

*s=n+f1(n-1);*

*return (s);*

*}*

1

**int(n==1)**

**return 1;**

**s=n+ f1(n-1);**

**return (s);**

1

n

s

int f1(int n)

**Dry Run:**

**K=f1(3)**

**3+f1(2)**

**2+f1(1)**

**How to approach a Recursive Problem?**

Step 1: Think function has already made.

Step 2: Recursive Case simpler version of original function.

Step 3: Base Case condition in which function will not be called.