Volume 5, No. 2, February 2014
Journal of Global Research in Computer Science
ISSN-2229-371X
REVIEW ARTICLE
Available Online at www.jgrcs.info

MATHEMATICAL DESCRIPTION OF VARIABLES, POINTERS, STRUCTURES, UNIONS USED IN C-TYPE LANGUAGE

Manoj Kumar Srivastava*, Asoke Nath
Indira Gandhi National Open University*, Department of Computer Science
St. Xavier's College (Autonomous) Kolkata India
*mksrivastav2011@rediffmail, asokejoy1@gmail.com

Abstract: Nath et al already have published a paper on mathematical description of keywords, variable declarations, arrays, user defined functions of c-type language. The authors have given the definition of variables, array initialization, calling a function from simple mathematical derivations or mathematical models. The authors have already shown that it is possible to conceptualize any high level language from mathematical derivations or mathematical models. In the present paper the authors tried to explore some of the important concepts of c-type language such as Pointers, structures, unions and variable declarations in c-type language. The structure is a complex data type and hence the authors tried to apply some simple mathematical logic or model to explain structure. Pointer is also one important concept in c-language and the authors have tried to explain it from simple mathematical models. The authors tried to establish that there is one to one mathematical logic or model for each component of any high level language. The present method may be further extended to other high level language, scripting language or object oriented language, database management systems etc.

Keywords: [c-type language , pointer, structure, mathematical logic]

INTRODUCTION

The language 'C' comprises of variables, data types, control statements, loop statements, subscripted variable of arrays, functions, strings, files, pointers, structures, unions, bit field, bit-wise operators, c-pre processor directives, c-unix interface etc. Nath et al [1] tried to explain mathematically the meaning of different data types, different control statements, arrays with some specific example. In the present paper the authors tried to explore the idea of pointers, structures , unions and variables using simple mathematical models. The same idea may be extended in future to design any new high level language. In C-language pointers means a special type of variables which can store the address of another variable. Structure is a derived data type where one can define various data types in the same structure. Union is a type where different data types will share the same memory locations. It is bit complex process as different data type will read data from same location. In present paper the authors tried to explore several common features such as control statements, loop statements, input and output functions or statement, relational operators, logical operators, different data types, arrays, pointer type variable, string operations, functions, subprograms or subroutines etc. In object oriented language there are something extra such as class, object, inheritance, polymorphism, data encapsulation etc.

The syntax of a particular statement may be different in two different languages but the purpose of the statement is the same. The authors tried to explain the different statements using some unified mathematical model. In the present paper the authors considered pointers, structures, unions of c language using simple mathematical model. The mathematical modeling of programming language such as C-language will help the user to understand a high level language much better. The authors also propose that the similar mathematical description can be applied to any other new language with minor revisions.

DIFFERENT MATHEMATICAL MODELS

In mathematics, a variable is a symbol designated a value that may change within the scope of a given set of problem or set of operation.

A variable is simply a symbol which is or may be associated to some value(s).

Variables are further divided into two categories:
(i) Dependent Variable
(ii) Independent Variable.

Independent Variables are regarded as input to a system and may take on different value freely.

Dependent Variables are those value that changes as a consequence of changes in other values in the systems.

A variable is just a named area of storage that can hold a single value(numeric or character).The C language demands that you declare the name of each variable that you are going to use its type.

The general symbol used to represent a variable is x, y, z, p, q, r…….etc. Now we are using a special symbol to represent the variable in the form *x,*y,*z,*i,*j,*k,…………………..etc. The difference between the symbol x and *x is that first x is used to represent address of a variable which points to some definite type of data and *x is the value where 'x' is pointing.

a. ***Pointer:*** A pointer is a variable that stores memory address. Like all other variable it also has a name , has to be declared and occupies some fixed space in memory . It is called pointer because it points to a particular location in memory by storing the address of that location.

To represent pointer in mathematical term we are taking two sets as follows :

S = { v ∈ Data : v contains the value of the variable}
X = { a ∈ Data : a contains the address of the variable}
we are taking the mapping T:S → X such that T(v)=a for v ∈ S and a ∈ X.

In the programming language C, a pointer is a variable itself; which stores the address of another variable of some specific type. Address of a variable is given by ampersand notation(&),which is called the unary operator or address operator that evaluates address of its operand. Here address of variable is dependent on the value of the address.
Example:

```
#include<stdio.h>
void main()
{
int    *a,  *b,  *c          /*Here a,b,c are three pointer pointing to some integer data */
int i=10, j=20,k=30;
a=&i ; / *a=Address of i*/
b=&j ;  /*b=Address of j */
c=&k ; /*c=Address of k*/
clrscr();
printf("Address of i=%x value of i=%d\n",a,*a);
printf("Address of j=%x value of j=%d\n",b,*b);
printf("Address of k=%x value of k=%d\n",c,*c);
getch();
}
```

If we take the above example in mathematical form it will be as follows: We take a mapping
T: S→X where
S={i=10,j=20,k=30} and
X={a,b,c}
T(i)=a
=&i;
T(j)=b
=&j;
and T(k)=c
=&k; The above mapping is one-one and onto also.

Properties for memory organization for pointer variable:

(i) When we use variable in a program then compiler keeps some memory for that variable depending on that data type.
(ii) The address given to that variable is unique with that variable name.
(iii) When Program execution starts the variable name is automatically translated into the corresponding address.

Now, keeping in mind the above properties if we define the mapping

T:S -> X such that T(v)=a for v ∈ S and a ∈ X. is an injective mapping. Where

S = { v ∈ Data : v contains the value of the variable}
X = { a ∈ Data : a contains the address of the variable}

Now we have to check whether the mapping T is injective or surjective:

The mapping T is injective because different value of variable have unique address of the variable.

The mapping T is not surjective because for each address there may not exit some value in domain set such that T(i)=j.

NOTE: Since the function T may or may not be surjective . Therefore inverse of T does not exit. Therefore from address of the variable we can not get the value of the variable.

NOTE :T(x+y) = T(x)+T(y) and T(cx)=cT(x) does not hold always because T is not surjective and therefore T is not bijective. So

a. addition of two pointers and multiplication by a constant should be avoided.
b. We cannot use Address operator for accessing address of literals
c. Output of (a+b) in a programming process is nothing but literals so we cannot use address Operator &(a+b).

The following result does not hold due to above mapping:
One can perform different arithmetic operation on pointer such as increment, decrement but still we have some more arithmetic operation that cannot be performed on C.

a. Addition of two addresses is not valid.
b. Multiplying two addresses is not valid.
c. Division of two addresses is not valid.
d. Modulo operation on pointers is not valid.
e. Bitwise AND ,OR ,XOR operation on pointer(s) is not valid .
f. NOT operation or negation operation on pointer is not possible.

Now the above points will be discussed programmatically :
1.Additon of two pointers:

```
#include<stdio.h>
int main()
{
int var=10;
int *ptr=&i;
int *ptr2=(int*)2000;
printf('%d",ptr1+ptr2);
return 0;
}
```

Output :
Compile error
2. Multiplication of pointer and a number:

```
#include<stdio.h>
int main()
{
int var=10;
int *ptr=&i;
int *ptr2=(int*)2000;
printf('%d",ptr1*var);
return 0;
}
```

Output :
Compile error

3. Multiplication of two pointer:

```
#include<stdio.h>
int main()
{
int var=10;
int *ptr=&i;
int *ptr2=(int*)2000;
printf('%d",ptr1*ptr2);
return 0;
}
```

Output :
Compile error

4. Modulo operation of two pointer:

```
#include<stdio.h>
int main()
{
int var=10;
int *ptr=(int*)1000;
int *ptr2=(int*)2000;
printf('%d",ptr2%ptr2);
return 0;
}
```

Output :
Compile error

5. Division of pointer:

```
#include<stdio.h>
int main()
{
int *ptr1,*ptr2;
int *ptr=(int*)1000;
ptr2=ptr1/4;
return0;
}
```

Output :
Illegal use of pointer

6. Can not perform bitwise OR

```
#include<stdio.h>
int main()
{
int  i=5,j=10;
int *p=&i;
int *q=&j;
printf('%d",ptr1|ptr2);
return0;
}
```

Output :
Compile error

7. Negation operator of two pointer:

```
#include<stdio.h>
int main()
{
int i=10;
int *ptr=&i;
printf('%d",~ptr);
return0;
}
```

Output :
Compile error

Summary at a glance:

```
Address   +   Address=  Illegal
Address   *   Address=  Illegal
Address   /   Address=  Illegal
Address   %   Address=  Illegal
Address   &   Address=  Illegal
Address   |   Address=  Illegal
Address   ^   Address=  Illegal
~Address                =Illegal
```

Subtraction operation in Pointer:

How subtraction properties hold on pointer? The mapping defined above is an injective mapping.so nearly Isometry rule hold between two sets with little change in the formula due to size of variable.

Computation of ptr2-ptr1 is done by the following formula;
Final result=(ptr2-ptr1)/size of data types

Pointer to Pointer:

A pointer to a variable ,is a variable ,that stores the address of another variable of specific type .A pointer to pointer is a variable ,that stores address of a pointer variable of specific type. The mapping between T: X $\rightarrow$ Y is given by

T(&a)=&(&a).

Where X = {a $\in$ Data : a have the value of the variable and &a is the address of the corresponding variable}
Y = {&a $\in$ Data : & a have the address of the variable}
Example

Example: Write a program to show pointer to another pointer :

```
#include<stdio.h>
void main()
{
int *a, **b, ***C ,****d;
int n=10
clrscr();
a=&n;
b=&a;
c=&b;
d=&c;
printf("Address of n=%x Value of n=%d\n",a,*a);
printf("Address of a=%x Address of n=%x Value of n=%d\n",b,*b,**b);
printf("Address of b=%x Address of a=%x Address of n=%x Value of n=%d\n",c,*c,**c,***c);
printf("  Address of c=%xAddress of b=%x Address of a=%x Address of n=%x Valueof n=%d\n",d,*d,**d,***d,****d);
getch();
}
```

Pointer to Array: The mapping defined between two set S={n[i] :i$\in$ N$\cup\{0\}$} and X={a[i] :i$\in$ N $\cup\{0\}$} is called the pointer to array

T(n[i])=&n[i]
=a[i]

which is equivalent to the mapping T:S$\rightarrow$ X
T(*(pt+i))=pt+i for i=0,1,2,……..,m
if we take the set S={*(pt+i) :pt is pointer variable and i=0,1,2,……,m} and
X= {(pt+i) : pt is pointer variable and i=0,1,2,……,m}

Example:Suppose we declare an array x as follows:
Int x[5]={1,2,3,4,5};
Suppose the base address of x is 1000 and assuming that each integer requires two bytes,the five elements will be stored as follows:

Table:1

Elements	x[0]	x[1]	x[2]	x[3]	x[4]
Value	1	2	3	4	5
Address	1000	1002	1004	1006	1008
	Base address				

The name x is defined as a constant pointer pointing to the first element,x[0] and therefore the value of x is 1000,the location where x[0] is stored. That is ,
x=&x[0]=1000

If we declare pt as an integer pointer, then we can make the pointer pt to point to the array x by the following assignment:
pt=x;

This is equivalent to pt=&x[0];

Now, We can access every value of x using pt++ to move from one element to another .The relationship between pt and x is shown as:

Table: 2

Pt	&x[0]	1000
pt+1	&x[1]	1002
pt+2	&x[2]	1004
pt+3	&x[3]	1006
pt+4	&x[4]	1008

Array of Pointers: Array of pointers is a collection of specific location of the memory from where one can get further address locations.
Example : int *a[4], *b[5];
Here *a[0], *a[1],*a[2],*a[3], *b[0],*b[1], *b[2], *b[3], *b[4] are the different locations in the memory from where we can reserve further memory addresses. Each element of the array is considered as pointer.
Structure : The general syntax of structure tag is
struct tag{
member1;
member2;
……………….
……………….
member n;
}var1,var2,var3,……..var n;
struct tag={member1,member2,member3 ,member4………..,member n}
={(member1,member2,member3……………..,member)∈ Data1xData2x……………xData n :each member is accessed by variable vi of given main() function is accessed in the form vi.memberj or vi->member or &vi.memberj for j=1,2,…….n and i=1,2,………}
Structure in structure:If S1 is subset of S2 then for each x∈ S1 => x∈ S2.Using this property we can prepare the conception of structrure in structure.

Struct tag1={(member1,member2,………….member n}
Struct tag2={{struct tag},membern+1,member n+2,……………,member p}
={member1,member2,member3,………………..,member n,member n+1,member n+2,………..member p}

Array of Structures:

All the data items in an array have the same name. Members of an array one-to-one to correspondence with set of positive integers including zero. Since array consists of finite number of terms. Hence they are connectable. The diagram for array in the mathematical way:

Variable

1	V1
2	V2
……..	…….
n-1	V(n-1)
N	Vn

Figure-3: Variable as Function

Variable [1] = Value 1
Variable [2] = Value 2
……………………….
Variable [n] = Value n

Variable

1	Sruct1
2	Struct2
……..	…….
n-1	struct(n-1)
N	Struct n

Figure-3: Variable as Function

struct[1]={member1,member2,member3,member4,………., member n}
struct[2]={{member1,member2,member3,member4,………. ,member n}
struct[3]={member1,member2,member3,member4,………., member n}
struct[4]={member1,member2,member3,member4,………., member n}
……………………………………………………………………………
……………………………………………………………………………
struct[n]={member1,member2,member3,member4,………., member n}
Pointers to structure: The general syntax for pointer to structure in an example can be as follws:
struct emp {
…………….
……………..
}e1, *ptr_emp;
ptr_emp=&e1;

Now we consider a mapping T:S→A where S is a structure set and A is set containing address of the variable. Now if we using the definition of pointer in mathematical way as defined above is:
T(member)= &member

=a where member $\in$ S and a $\in$ A.

But in the pointer to structure we access the member as in the following way :

T(member)=&(*pointer.member)

Or

T(member)=&(pointer$\rightarrow$ member)

This result can be shown in this example as follows:

```
Struct name {
        member 1;
        member2;
………….
…………..
…………….
};
----------inside function------------
 struct name *ptr;
```

Here the pointer variable of type struct name is created structure member through pointer can be used in two ways.
a. Referencing pointer to another address to access memory.
b. Using dynamic memory allocation.

Example: Consider an example to access structure's member through pointer :

```
#include<stdio.h>
struct name{
  int a;
 float b;
}
int main()
{
struct name*ptr, p;
ptr=&p; /*referencing pointer to memory address of p */
printf("Enter integer :");
scanf('%d ",&(*ptr).a);
printf ("Enter number:");
scanf("%f",&(*ptr).b);
printf("Displaying :");
printf("%d%f",(*ptr).a,(*ptr).b);
return 0;
}
```

In this example ,the pointer variable of the type struct name is referenced to the address of p .Then only the structure member through pointer can be accessed.

Structure pointer member can be accessed using -> operator.

(*ptr).a is same as ptr->a
(*ptr).b is same as ptr->b

Accessing structure member through pointer using dynamic memory allocation:- To access structure member using pointer , memory can be allocated dynamically using malloc() function defined under "stdlib.h" header file.

Syntax of use of malloc() is:

```
  ptr=(cast_type*)malloc(byte_size)
```

Example to use Structure's member through pointer using malloc() function:

```
#include<stdio.h>
#include<stdlib.h>
struct name{
int a;
float b;
char c[30];
}
int main()
{
struct name *ptr;
int i,n;
 printf('Enter n:");
scanf("%d",&n);
ptr=(struct name*)malloc(n*size of (struct name));
/*above statement allocates the memory for on structure with  pointer ptr pointing to base address*/
for(i=0;i<n; i++)
printf("Enter string,integer and floating number respectively:\n");
scanf("%s%d%f",&(ptr+i)->c,&(ptr+i)->b,&(ptr+i)->b);
}
printf("Displaying information;\n");
for(i=0;i<n;i++)
printf("%s\t%d\t%.2f\n",(ptr+i)->c,(ptr+i)->a,(ptr+i)->b);
return 0;
}
```

2.8 : Pointer within structure:

(i) Structure may contain the pointer variable as member.
(ii)Pointer are used to store address of memory location
(iii)They can be de-referenced by '*' operator.

The above case is defined for pointer within structure as follows

Struct tag={member1,member2,member3 ,member4………..,member n}
={(member1,member2,member3……………..,member)$\in$ Data1 x Data2 x……………x Data n: pointer member is accessed by value of variable vi of given main() function is accessed in the form vi.member j or vi->member or *vi.memberj or *vi->member j for j=1,2,…….n and i=1,2,……….}

Example:

```
#include<stdio.h>
struct student
{
int *ptr; /* store address of integer variable*/
char *name;/*store address of character string*/
}s1;
int main()
{
int roll=20;
s1.ptr=&roll;
s1.name="aayush";
printf("\n Roll number of student:%d",*s1.ptr);
printf("\n Name of student:%s",s1.name);
}
```

Union: The general syntax of union tag is

```
 union tag{
 member1;
 member2;
……………….
……………….
 member n;
}var1,var2,var3,……..var m;
```

Union tag={member1,member2,member3 ,member4………….,member n}
={(member1,member2,member3……………..,member)∈ Data1xData2x……………xData n :each member is accessed by variable vi of given main() function is accessed in the form vi.memberj or vi->member or &vi.member j for j=1,2,…….n and i=1,2,3……….m}

Note : A union is used for applications involving multiple members ,where values of only one member need to be used at one time.

Relational Operators :==,<,>,<=,>=,!= are follows due to properties of real numbers which are following:(i)Algebraic properties
(ii)ordered properties
(iii)Density properties
(iv)Completeness properties
(v)Archimedian properties

Logical Operator:- The properties of logical operator &&, ||, ! holds due to Boolean algebra.

2.12. gotoxy() function

gotoxy()={(row,col)∈ NxN :there is some properties of the variable at that position}

Here N is set of natural number.
:

CONCLUSION AND FUTURE SCOPE

In the present paper the authors have studied how some important statements of C-language can be verified or compared with various mathematical models or functions. In the present paper the authors have shown how pointers, structures can be compared with different simple mathematical description. The language used in this paper is C-language. The similar concept may be applied to object oriented language or any scripting language. In the present paper the authors have focused on statements of C-language, data types, storage allocation, pointers, structures. The similar mathematical analysis may be done for any other high level language specially object oriented language like Java, C# etc.

REFERENCES

[1]. Mathematical modeling of various statements of C-type Language, Manoj Kumar Srivastav , Asoke Nath, International Journal of Advanced Computer Research(IJACR), Vol-3,Number-1, Issue-13, Page:79-87 Dec(2013).

[2]. Real Analysis , S.K.Mapa, publication-Asoke Prakasan, year1998

[3]. Higher Algebra, S.K.Mapa,publication-Sarat Book Distribution,year 2000

[4]. Discerete Mathematics with Proof , Eric Gossett, Wiley , India

[5]. Discrete Mathematics, N.Chandrasekaran, M.Umaparvathi, PHI, 2010

[6]. Discrete Mathematics and its Applications, K.H.Rosen, Tata Mcgraw-Hill Publishing Company limited, 2003

[7]. Programming in ANSI C, E.Balagurusamy, Tata Mcgraw Hill Book Company, 2004

[8]. Calculus and analytic geometry-Thomas, Finney pearson eduation asia-2002

[9]. Mathematcs- Dr.R.D.Sharma-Dhanpat Rai publication (p)ltd-2010

[10]. notes of constructive mathematics and computer programming-P.Martin Lof.

[11]. complex variable and application-churchill Brown.McGraw-Hill International publication1996

[12]. www.c4learn.com