Pointer

{C}
Programming

What is Pointer?

- A normal variable is used to store value.
- A pointer is a variable that store address / reference of another variable.
- Pointer is derived data type in C language.
- A pointer contains the memory address of that variable as their value. Pointers are also called address variables because they contain the addresses of other variables.

Declaration & Initialization of Pointer

Syntax

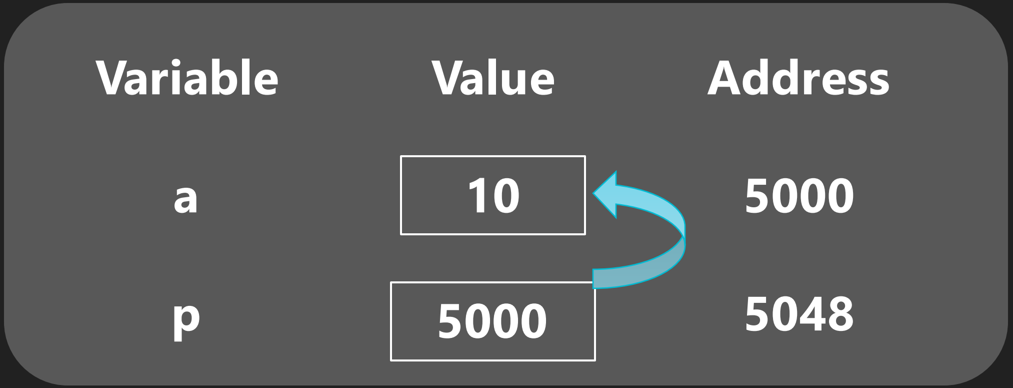
```
datatype *ptr_variablename;
```

Example

```
void main()
{
    int a=10, *p; // assign memory address of a
    to pointer variable p
    p = &a;
    printf("%d %d %d", a, *p, p);
}
```

Output

```
10 10 5000
```



Variable	Value	Address
a	10	5000
p	5000	5048

- p is integer pointer variable
- & is address of or referencing operator which returns memory address of variable.
- * is indirection or dereferencing operator which returns value stored at that memory address.
- & operator is the inverse of * operator
- x = a is same as x = *(&a)

Why use Pointer?

- C uses pointers to create dynamic data structures, data structures built up from blocks of memory allocated from the heap at run-time. Example linked list, tree, etc.
- C uses pointers to handle variable parameters passed to functions.
- Pointers in C provide an alternative way to access information stored in arrays.
- Pointer use in system level programming where memory addresses are useful. For example shared memory used by multiple threads.
- Pointers are used for file handling.
- This is the reason why C is versatile.

Pointer to Pointer – Double Pointer

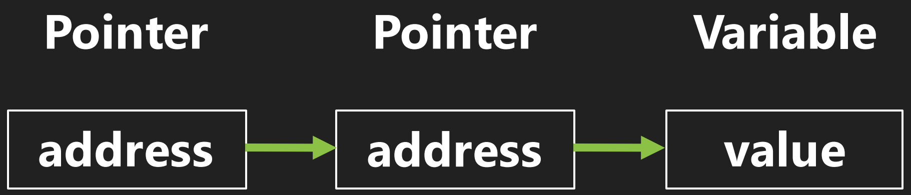
- Pointer holds the address of another variable of same type.
- When a pointer holds the address of another pointer then such type of pointer is known as pointer-to-pointer or double pointer.
- The first pointer contains the address of the second pointer, which points to the location that contains the actual value.

Syntax

```
datatype **ptr_variablename;
```

Example

```
int **ptr;
```



Write a program to print variable, address of pointer variable and pointer to pointer variable.

Program

```c
#include <stdio.h>
int main () {
    int var;
    int *ptr;
    int **pptr;
    var = 3000;
    ptr = &var; // address of var
    pptr = &ptr; // address of ptr using address of operator &
    printf("Value of var = %d\n", var );
    printf("Value available at *ptr = %d\n", *ptr );
        printf("Value available at **pptr = %d\n", **pptr);
        return 0;
}
```

Output

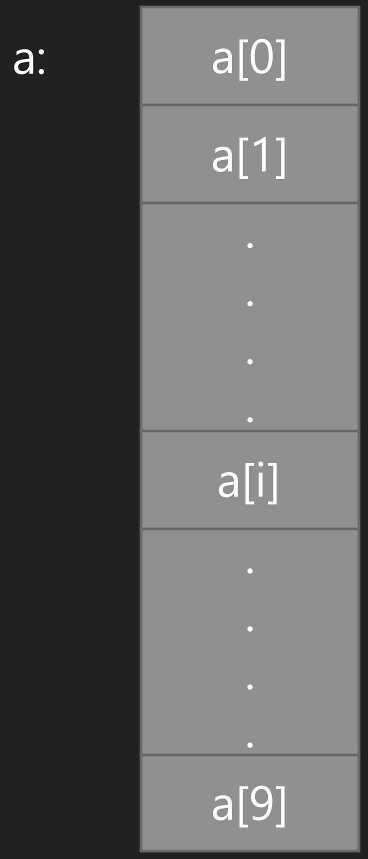
```
Value of var = 3000
Value available at *ptr = 3000
Value available at **pptr = 3000
```

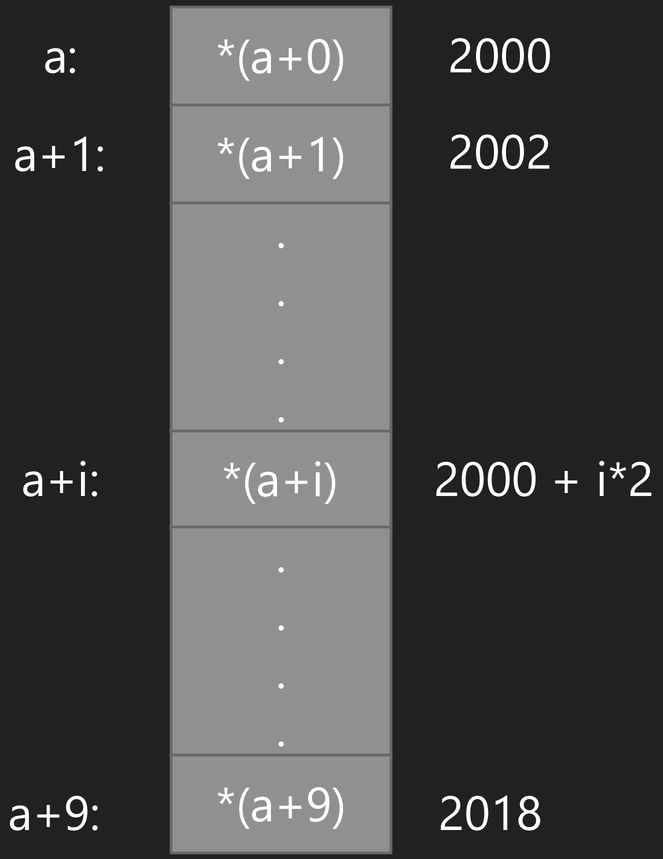
Relation between Array & Pointer

- When we declare an array, compiler allocates continuous blocks of memory so that all the elements of an array can be stored in that memory.
- The address of first allocated byte or the address of first element is assigned to an array name.
- Thus array name works as pointer variable.
- The address of first element is also known as base address.

Relation between Array & Pointer – Cont.

- Example: `int a[10], *p;`
- a[0] is same as *(a+0), a[2] is same as *(a+2) and a[i] is same as *(a+i)





Array of Pointer

- As we have an array of char, int, float etc, same way we can have an array of pointer.
- Individual elements of an array will store the address values.
- So, an array is a collection of values of similar type. It can also be a collection of references of similar type known by single name.

Syntax

```
datatype *name[size];
```

Example

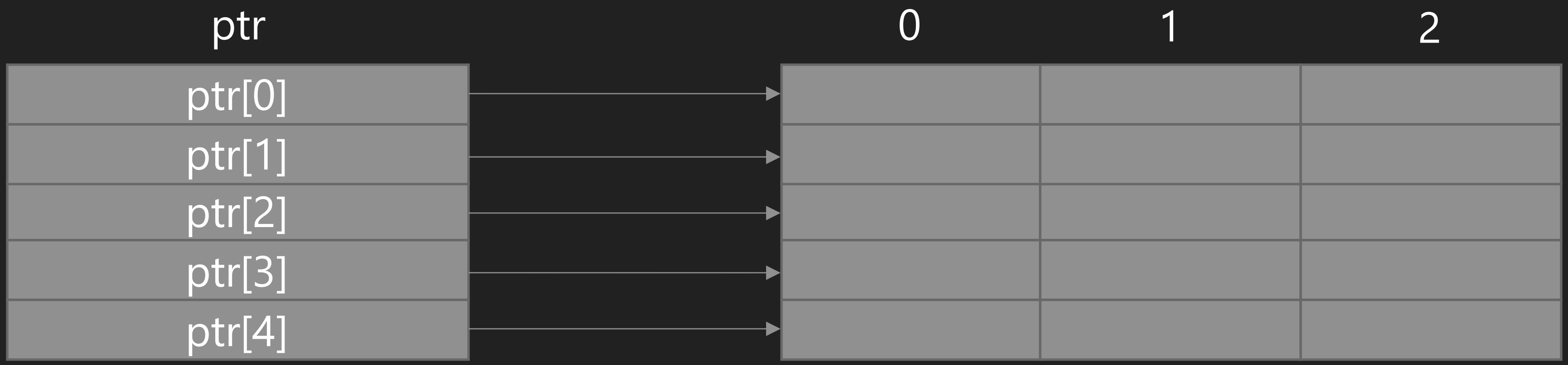
```
int *ptr[5]; //declares an array of integer pointer of size 5
```

Array of Pointer – Cont.

- An array of pointers ptr can be used to point to different rows of matrix as follow:

Example

```
for(i=0; i<5; i++)
{
        ptr[i]=&mat[i][0];
}
```



- By dynamic memory allocation, we do not require to declare two-dimensional array, it can be created dynamically using array of pointers.

Write a program to swap value of two variables using pointer / call by reference.

Program

```c
int main()
{
    int num1,num2;
    printf("Enter value of num1 and num2: ");
    scanf("%d %d",&num1, &num2);

//displaying numbers before swapping
    printf("Before Swapping: num1 is: %d, num2 is: %d\n",num1,num2);

//calling the user defined function swap()
    swap(&num1,&num2);

//displaying numbers after swapping
    printf("After Swapping: num1 is: %d, num2 is: %d\n",num1,num2);
    return 0;
}
```

Output

```
Enter value of num1 and num2: 5
10
Before Swapping: num1 is: 5, num2 is: 10
After  Swapping: num1 is: 10, num2 is: 5
```

Pointer and Function

- Like normal variable, pointer variable can be passed as function argument and function can return pointer as well.
- There are two approaches to passing argument to a function:
  - Call by value
  - Call by reference / address

Call by Value

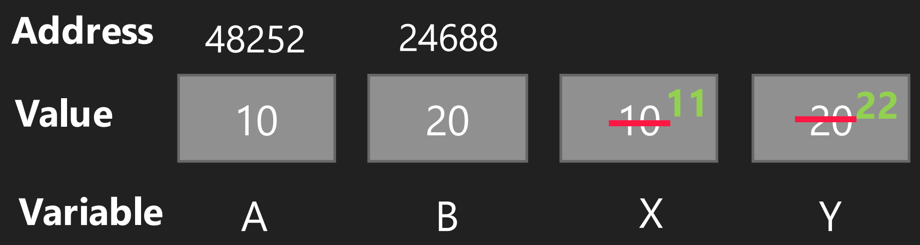
- In this approach, the values are passed as function argument to the definition of function.

Program

```c
#include<stdio.h>
void fun(int,int);
int main()
{
     int A=10,B=20;
     printf("\nValues before calling %d, %d",A,B);
     fun(A,B);
     printf("\nValues after calling %d, %d",A,B);
     return 0;
}
void fun(int X,int Y)
{
     X=11;
     Y=22;
}
```

Output

```
Values before calling 10, 20
Values after calling 10, 20
```



Call by Reference / Address

- In this approach, the references / addresses are passed as function argument to the definition of function.

Program

```c
#include<stdio.h>
void fun(int*,int*);
int main()
{
    int A=10,B=20;
    printf("\nValues before calling %d, %d",A,B);
    fun(&A,&B);
    printf("\nValues after calling %d, %d",A,B);
    return 0;
}
void fun(int *X,int *Y)
{
    *X=11;
    *Y=22;
}
```

Output

```
Values before calling 10, 20
Values after  calling 11, 22
```

Address	48252	24688		
Value	~~10~~ 11	~~20~~ 22	48252	24688
Variable	A	B	*X	*Y

Pointer to Function

- Every function has reference or address, and if we know the reference or address of function, we can access the function using its reference or address.
- This is the way of accessing function using pointer.

Syntax

```
return-type (*ptr-function)(argument list);
```

- return-type: Type of value function will return.
- argument list: Represents the type and number of value function will take, values are sent by the calling statement.
- (*ptr-function): The parentheses around *ptr-function tells the compiler that it is pointer to function.
- If we write *ptr-function without parentheses then it tells the compiler that ptr-function is a function that will return a pointer.

Write a program to sum of two numbers using pointer to function.

Program

```c
#include<stdio.h>
int Sum(int,int);
int (*ptr)(int,int);
int main()
{
    int a,b,rt;
    printf("\nEnter 1st number : ");
    scanf("%d",&a);
    printf("\nEnter 2nd number : ");
    scanf("%d",&b);
    ptr = Sum;
    rt = (*ptr)(a,b);
    printf("\nThe sum is : %d",rt);
    return 0;
}
int Sum(int x,int y)
{
        return x + y;
}
```

Output

```
Enter 1st number : 5

Enter 2nd number : 10

The sum is : 15
```

Practice Programs

1. Write a C program to print the address of variable using pointer.
2. Write a C a program to swap two elements using pointer.
3. Write a C a program to print value and address of a variable
4. Write a C a program to calculate sum of two numbers using pointer
5. Write a C a program to swap value of two numbers using pointer
6. Write a C a program to calculate sum of elements of an array using pointer
7. Write a C a program to swap value of two variables using function
8. Write a C a program to print the address of character and the character of string using pointer
9. Write a C a program for sorting using pointer

Thank you