VTC Academy
Think Ahead, Beyond Boundaries

BASIC PROGRAMMING LANGUAGE

LESSON 9

Pointers

VTC Academy

@ 2021 VTC Academy

CONTENT

VTC Academy
Think Ahead, Beyond Boundaries

VTC Academy

@ 2021 VTC Academy

Address in C

- If you have a variable var in your program, &var will give you its address in the memory.
- We have used address numerous times while using the `scanf()` function.

```
scanf("%d", &var);
```

@ 2021 VTC Academy

Variable Address Example

VTC Academy
Think Ahead, Beyond Boundaries

```c
#include <stdio.h>
int main()
{
    int var = 5;
    printf("var: %d\n", var);
    printf("Address of var: %p", &var);
    printf("\n");
    return 0;
}
```

VTC Academy

@ 2021 VTC Academy

What are Pointers?

VTC Academy
Think Ahead, Beyond Boundaries

- A pointer is a variable, which contains the address of a memory location of another variable.
- If one variable contains the address of another variable, the first variable is said to point to the second variable.
- A pointer provides an indirect method of accessing the value of a data item.
- Pointers can point to variables of other fundamental data types like `int`, `char`, or `double` or data aggregates like arrays or structures.

VTC Academy

@ 2021 VTC Academy

Pointer Variables

VTC Academy
Think Ahead, Beyond Boundaries

- A pointer declaration consists of a base type and a variable name preceded by symbol (*).
- General declaration syntax is :

```
type *name;
```

- For example:

```
int *ptr;
```

VTC Academy

@ 2021 VTC Academy

Pointer Operators

VTC Academy
Think Ahead, Beyond Boundaries

- There are 2 special operators which are used with pointers & and *.
- Operator & is a unary operator and it returns the memory address of the operand.

```
ptr = &var;
```

- Operator * is the complement of &, returns the value contained in the memory location pointed to by the pointer variable's value.

```
value = *ptr;
```

VTC Academy

@ 2021 VTC Academy

Pointer Example

```c
#include <stdio.h>
int main() {
    int *pc, c;
    c = 22;
    printf("Address of c: %p\n", &c);
    printf("Value of c: %d\n\n", c);
    pc = &c;
    printf("Address of pointer pc: %p\n", pc);
    printf("Content of pointer pc: %d\n\n", *pc);
    c = 11;
    printf("Address of pointer pc: %p\n", pc);
    printf("Content of pointer pc: %d\n\n", *pc);
    *pc = 2;
    printf("Address of c: %p\n", &c);
    printf("Value of c: %d\n\n", c);
    return 0;
}
```

VTC Academy

@ 2021 VTC Academy

Pointer Arithmetic

VTC Academy Think Ahead, Beyond Boundaries

- Addition and subtraction are the only operations that can be performed on pointers.

```
int var, *ptr;
ptr = &var;
var = 500;
ptr++;
```

- Let us assume that `var` is stored at the address 1000.
- Then `ptr` has the value 1000 stored in it. Since integers are 2 bytes long, after the expression "`ptr++;`" `ptr` will have the value as 1004 and not 1001.

VTC Academy

@ 2021 VTC Academy

Pointer Arithmetic

VTC Academy
Think Ahead, Beyond Boundaries

- Each time a pointer is incremented, it points to the memory location of the next element of its base type.
- Each time it is decremented it points to the location of the previous element.
- All other pointers will increase or decrease depending on the length of the data type they are pointing to.

VTC Academy

@ 2021 VTC Academy

Pointer Arithmetic

VTC Academy
Think Ahead, Beyond Boundaries

Pointer Arithmetic	Description
`++ptr_var or ptr_var++`	Points to next integer after var
`–ptr_var or ptr_var–`	Points to integer previous to var
`ptr_var +1`	Points to the ith integer after var
`ptr_var –1`	Points to the ith integer before var
`++*ptr_var or (*ptr_var)++`	Will increment var by 1
`*ptr_var++`	Will fetch the value of the next integer after var

VTC Academy

@ 2021 VTC Academy

Pointer Comparisons

VTC Academy
Think Ahead, Beyond Boundaries

- Two pointers can be compared in a relational expression provided both the pointers are pointing to variables of the same type.
- Pointers of the same type (after pointer conversions) can be compared for equality. Two pointers of the same type compare equal if and only if they are both null, both point to the same function, or both represent the same address.
- Consider that ptr_a and ptr_b are 2 pointer variables, which point to data elements a and b. In this case the following comparisons are possible:

VTC Academy

@ 2021 VTC Academy

Pointer Comparisons

VTC Academy
Think Ahead, Beyond Boundaries

`ptr_a < ptr_b`	Returns true provided a is stored before b
`ptr_a > ptr_b`	Returns true provided a is stored after b
`ptr_a <= ptr_b`	Returns true provided a is stored before b or ptr_a and ptr_b point to the same location
`ptr_a >= ptr_b`	Returns true provided a is stored after b or ptr_a and ptr_b point to the same location.
`ptr_a = ptr_b`	Returns true provided both pointers ptr_a and ptr_b points to the same data element.
`ptr_a != ptr_b`	Returns true provided both pointers ptr_a and ptr_b point to different data elements but of the same type.
`ptr_a = NULL`	Returns true if ptr_ais assigned NULL value (zero)

@ 2021 VTC Academy

NULL Pointers

VTC Academy
Think Ahead, Beyond Boundaries

- It is always a good practice to assign a NULL value to a pointer variable in case you do not have an exact address to be assigned. This is done at the time of variable declaration.
- A pointer that is assigned NULL is called a null pointer.
- The NULL pointer is a constant with a value of zero defined in several standard libraries.
- For example:

```
int *ptr = NULL;
```

VTC Academy

@ 2021 VTC Academy

Pointer & Single Dimensional Arrays

VTC Academy
Think Ahead, Beyond Boundaries

- In most contexts, array names decay to pointers. In simple words, array names are converted to pointers.
- That's the reason why you can use pointers to access elements of arrays. However, you should remember that pointers and arrays are not the same.
- The address of an array element can be expressed in two ways :
  - By writing the actual array element preceded by the ampersand sign (&)
  - By writing an expression in which the subscript is added to the array name

VTC Academy

@ 2021 VTC Academy

Pointer & Single Dimensional Arrays

VTC Academy
Think Ahead, Beyond Boundaries

```c
#include <stdio.h>

int main()
{
    int ary[10] = {10, 21, 32, 43, 54, 65, 76, 87, 98, 99};
    int i;
    for (i = 0; i < 10; i ++)
    {
        printf("\ni=%d,ary[i]=%d,*(ary+i)=%d", i, ary[i], *(ary + i));
        printf("&ary[i]=%X,ary+i=%X", &ary[i],ary+i);
    }

    return 0;
}
```

VTC Academy

@ 2021 VTC Academy

Pointer & Multi Dimensional Arrays

VTC Academy
Think Ahead, Beyond Boundaries

- A two-dimensional array can be defined as a pointer to a group of contiguous one-dimensional arrays.
- A two-dimensional array declaration can be written as:

```
data_type (*ptr_var) [expr 2];
```

- instead of

```
data_type ptr_var [expr1][expr 2];
```

VTC Academy

@ 2021 VTC Academy

Dynamic Memory Allocation

VTC Academy
Think Ahead, Beyond Boundaries

- As you know, an array is a collection of a fixed number of values. Once the size of an array is declared, you cannot change it.
- Sometimes the size of the array you declared may be insufficient. To solve this issue, you can allocate memory manually during run-time. This is known as dynamic memory allocation in C programming.
- The C programming language provides several functions for memory allocation and management.

@ 2021 VTC Academy

Dynamic Memory Allocation

VTC Academy Think Ahead, Beyond Boundaries

- These functions can be found in the `<stdlib.h>` header file:
  - `void *malloc(int num);`
    This function allocates an array of num bytes and leave them initialized
  - `void *calloc(int num, int size);`
    This function allocates an array of num elements each of which size in bytes will be size
  - `void *realloc(void *address, int newsize);`
    This function re-allocates memory extending it upto newsize
  - `void free(void *address);`
    This function releases a block of memory block specified by address

@ 2021 VTC Academy

malloc() Function Demo

VTC Academy
Think Ahead, Beyond Boundaries

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int *a;
    int count, i;

    printf("Input array size: ");
    scanf("%d", &count);

    //alloc memory
    a = (int*)malloc(count*sizeof(int));

    //input array:
    for(i=0; i<count; i++){
        printf("a[%d] = ", i);
        scanf("%d", a + i);
    }

    //display array:
    printf("\nArray:\n");
    for(i=0; i<count; i++){
        printf("%8d", a[i]);
    }

    //free memory
    free(a);

    return 0;
}
```

VTC Academy

@ 2021 VTC Academy

calloc() Function Demo

VTC Academy
Think Ahead, Beyond Boundaries

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int *a;
    int count, i;

    printf("Input array size: ");
    scanf("%d", &count);

    //alloc memory
    a = (int*)calloc(count,sizeof(int));

    //input array:
    for(i=0; i<count; i++){
        printf("a[%d] = ", i);
        scanf("%d", a + i);
    }

    //display array:
    printf("\nArray:\n");
    for(i=0; i<count; i++){
        printf("%8d", a[i]);
    }

    //free memory
    free(a);

    return 0;
}
```

VTC Academy

@ 2021 VTC Academy

realloc() Function Demo

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int *a;
    int count, i;

    printf("Input array size: ");
    scanf("%d", &count);

    //alloc memory
    a = (int*)calloc(count,sizeof(int));

    //input array:
    for(i=0; i<count; i++){
        printf("a[%d] = ", i);
        scanf("%d", a + i);
    }

    //realloc memory
    a = (int*)realloc(a, (count+1)*sizeof(int));
    printf("Input new element: ");
    scanf("%d", &a[count]);
    count++;

    //display array:
    printf("\nArray:\n");
    for(i=0; i<count; i++){
        printf("%8d", a[i]);
    }

    //free memory
    free(a);

    return 0;
}
```

VTC Academy

@ 2021 VTC Academy

Pointers and Functions

- In C programming, it is also possible to pass addresses as arguments to functions.
  - Call by Value
  - Call by Reference
- Function Pointers

VTC Academy

@ 2021 VTC Academy

Call By Value

VTC Academy
Think Ahead, Beyond Boundaries

```c
#include <stdio.h>

/* function declaration */
void swap(int x, int y);

int main () {

   /* local variable definition */
   int a = 100;
   int b = 200;

   printf("Before swap, value of a : %d\n", a );
   printf("Before swap, value of b : %d\n", b );

   /* calling a function to swap the values */
   swap(a, b);

   printf("After swap, value of a : %d\n", a );
   printf("After swap, value of b : %d\n", b );

   return 0;
}
```

```c
/* function definition to swap the values */
void swap(int x, int y) {

   int temp;

   temp = x; /* save the value of x */
   x = y;    /* put y into x */
   y = temp; /* put temp into y */

   return;
}
```

VTC Academy

@ 2021 VTC Academy

Call By Reference

VTC Academy
Think Ahead, Beyond Boundaries

```c
#include <stdio.h>

/* function declaration */
void swap(int *x, int *y);

int main () {

   /* local variable definition */
   int a = 100;
   int b = 200;

   printf("Before swap, value of a : %d\n", a );
   printf("Before swap, value of b : %d\n", b );

   /* calling a function to swap the values.
      &a indicates pointer to a ie. address of variable a
      &b indicates pointer to b ie. address of variable b
   */
   swap(&a, &b);

   printf("After swap, value of a : %d\n", a );
   printf("After swap, value of b : %d\n", b );

   return 0;
}
```

```c
/* function definition to swap the values */
void swap(int *x, int *y) {

   int temp;
   temp = *x;    /* save the value at address x */
   *x = *y;      /* put y into x */
   *y = temp;    /* put temp into y */

   return;
}
```

VTC Academy

@ 2021 VTC Academy

Function Pointers

VTC Academy Think Ahead, Beyond Boundaries

- Function Pointers point to code like normal pointers. In Functions Pointers, function's name can be used to get function's address.
- A function can also be passed as an arguments and can be returned from a function.
- By using function pointers, a function can be sent as a parameter to another function.
- This feature enables the C program to load function dynamically at runtime.
- Syntax:

```
function_return_type(*Pointer_name)(function argument list)
```


@ 2021 VTC Academy

Function Pointers Example

VTC Academy
Think Ahead, Beyond Boundaries

```c
#include <stdio.h>
int subtraction(int a, int b) {
    return (a - b);
}
int main() {
    int (*fp)(int, int) = subtraction;
    int result = fp(5, 4);
    printf("Result from function pointer: %d\n", result);
    return 0;
}
```

VTC Academy

@ 2021 VTC Academy

Summary

- A pointer is a variable, which contains the address of a memory location of another variable.
- If one variable contains the address of another variable, the first variable is said to point to the second variable.
- The C programming language provides several functions for memory allocation and management.
- To allocate memory dynamically, library functions are `malloc()`, `calloc()`, `realloc()` and `free()` are used. These functions are defined in the `<stdlib.h>` header file.
-

@ 2021 VTC Academy

VTC Academy
Think Ahead, Beyond Boundaries

VTC Academy

@ 2021 VTC Academy