KATHMANDU UNIVERSITY SCHOOL OF MANAGEMENT

BBIS

COM 102 : 3 Credit Hours

9. Pointers

Outlines

9. Pointers (6 hrs)

  9.1 Pointer Declaration

  9.2 Pointer Arithmetic

  9.3 Operation on Pointers

  9.4 Pointer and Array (Pointer and one dimension Array)

  9.5 Multidimensional

  9.5 Dynamic Memory Allocation

• Address in C

- Suppose, we have a variable named “var”, then its address can be accessed with a “&” by “&var”.
- Also, we have used this concept several times in scanf() function.
- As, scanf(“%d”, &var);

Example

```
#include <stdio.h>
int main()
{
  int var = 5;
  printf("var: %d\n", var);

  // Notice the use of & before var
  printf("address of var: %p", &var);
  return 0;
}
```

Output

```
var: 5
Address of var: Address may varies depending on your RAM.
```

C Pointers

- Pointers (pointer variables) are special variables that are used to store addresses rather than values.
- Indirect way of accessing and manipulating a variable content.
- This variable can be of type int, char, array, function, or any other pointer.
- The size of the pointer depends on the architecture.
- However, in 32-bit architecture the size of a pointer is 2 byte.

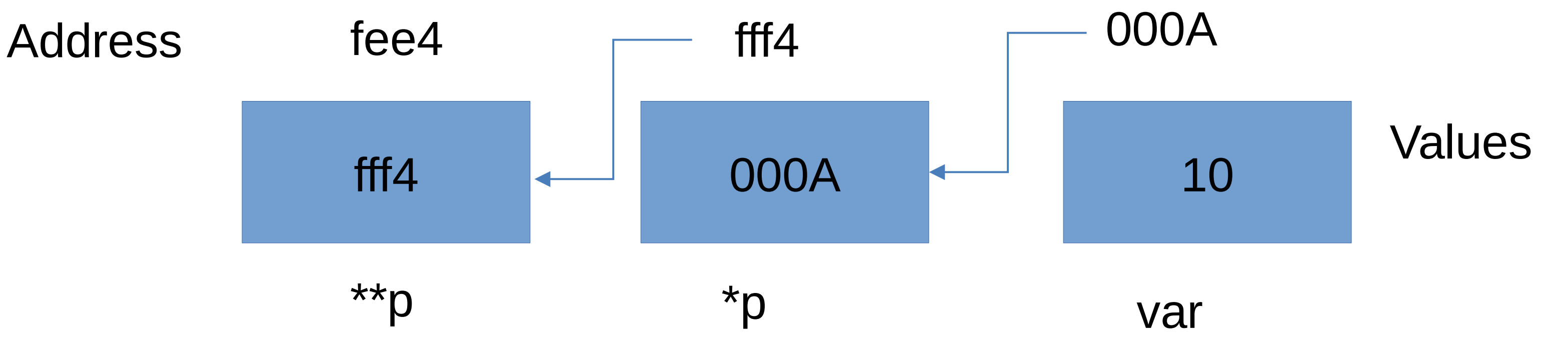
Pointer Syntax

- Here is how we can declare pointers.

```
int *p;
```

Here, a pointer “p” of int type is declared.

Example



Here, pointer variable stores the address of number variable, i.e., 000A.
The value of number variable is 10. But the address of pointer variable p is fff4.

• Advantage of pointer

1) Pointer reduces the code and improves the performance, it is used to retrieving strings, trees, etc. and used with arrays, structures, and functions.

2) We can return multiple values from a function using the pointer.

3) It makes you able to access any memory location in the computer's memory.

Usage of pointer

1) Dynamic memory allocation

    1)- dynamically allocate memory using malloc() and calloc() functions where the pointer is used.

2) Arrays, Functions, and Structures

    1)- used in arrays, functions, and structures. It reduces the code and improves the performance.

Contd...

- **You can also declare pointers in these ways.**

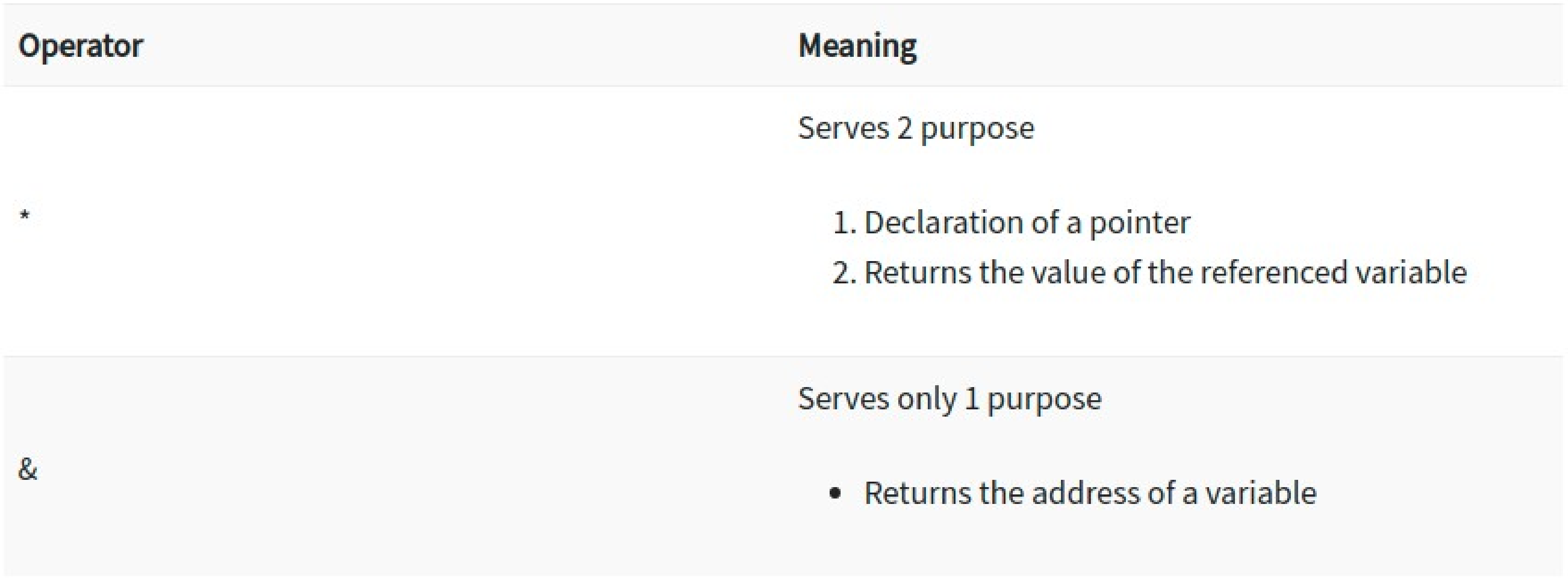
```
int *p1;
Int * p2;
```

- Let's take another example of declaring pointers.

```
int* p1, p2;
```

- Here, we have declared a pointer p1 and a normal variable p2.

Contd...



Operator	Meaning
*	Serves 2 purpose 1. Declaration of a pointer 2. Returns the value of the referenced variable
&	Serves only 1 purpose • Returns the address of a variable

Source: https://www.guru99.com/c-pointers.html

Contd...

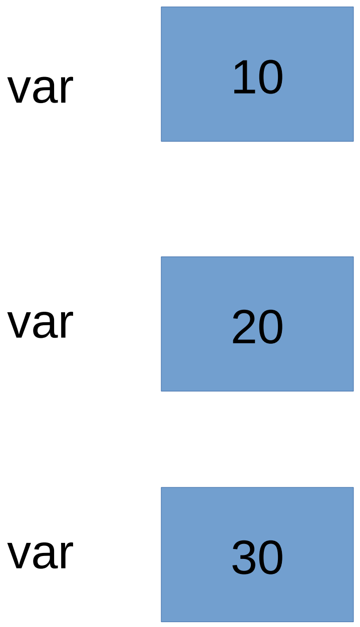
- Assigning addresses to Pointers
- Let's take an example.

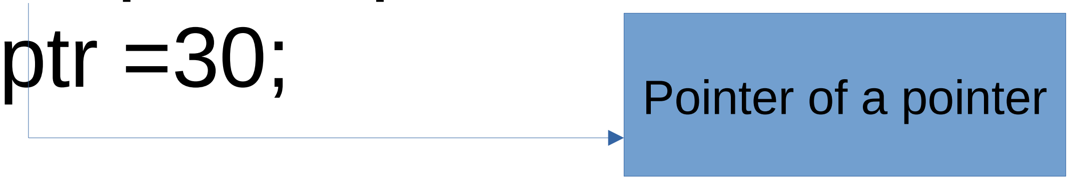
```
int* pc, c;
c = 5;
pc = &c;
```

- Here, 5 is assigned to the c variable. And, the address of c is assigned to the pc pointer.

How pointer works in C

```
int var = 10;

int *ptr = &var;
*ptr=10;
*ptr = 20;

Int **ptr = &ptr;
**ptr =30;
```





```
#include<stdio.h>
main()
{
int u=5;
int *pu;
printf("

\n u=%d &u=%x pu=%x

*pu=%d",u,&u,pu,*pu);
}
```

*pu and pu give garbage value

Get Value of Thing Pointed by Pointers

we use the * operator to get the value of the thing pointed by the pointers.
For example:

```
int* pc, c;
c = 5;
pc = &c;
printf("%d", *pc);   // Output: 5
```

In the above example, pc is a pointer, not *pc. You cannot and should not do something like *pc = &c;

* is called the dereference operator (when working with pointers). It operates on a pointer and gives the value stored in that pointer.

Changing Value Pointed by Pointers

```
int* pc, c;
c = 5;
pc = &c;
c = 1;
printf("%d", c);    // Output: 1
printf("%d", *pc);  // Ouptut: 1
```

Example

```
int* pc, c, d;
c = 5;
d = -15;
pc = &c;
printf("%d", *pc); // Output: 5
pc = &d;
printf("%d", *pc); // Ouptut: -15
```

Pointer Program to swap two numbers without using the 3rd variable.

```
int a=10,b=20,;
printf("Before swap: *p1=%d *p2=%d",*p1,*p2);
a=a+b;
b=a-b;
a=a-b;
printf("\nAfter swap: *p1=%d *p2=%d",*p1,*p2);
```

Pointer Arithmetic

- Increment (++)
- Decrement (--)
- an integer may be added to a pointer (+ or +=)
- an integer may be subtracted from a pointer (– or -=)
- Note: Pointer arithmetic is meaningless unless performed on an array.

Contd...

- Pointers contain addresses.
- Adding two addresses makes no sense,
- As, there is no idea what it would point to.
- Subtracting two addresses lets you compute the offset between these two addresses.

C++ program to illustrate Pointer Arithmetic

```
// Declare an array
    int v[3] = {10, 100, 200};
    // Declare pointer variable
    int *ptr;
    // Assign the address of v[0] to ptr
    ptr = v;
    for (int i = 0; i < 3; i++)
    {
        printf("Value of *ptr = %d\n", *ptr);
        printf("Value of ptr = %p\n\n", ptr);
        // Increment pointer ptr by 1
        ptr++;
    }
```

Operation	Explanation
Assignment	int *P1,*P2 P1=P2; P1 and P2 point to the same integer variable
Incrementation and decrementation	Int *P1; P1++;P1– ;
Adding an offset (Constant)	This allows the pointer to move N elements in a table. The pointer will be increased or decreased by N times the number of byte (s) of the type of the variable. P1+5;

Pointers & Arrays

- An array name acts like a pointer constant. The value of this pointer constant is the address of the first element.
- For example, if we have an array named val then val and &val[0] can be used interchangeably.

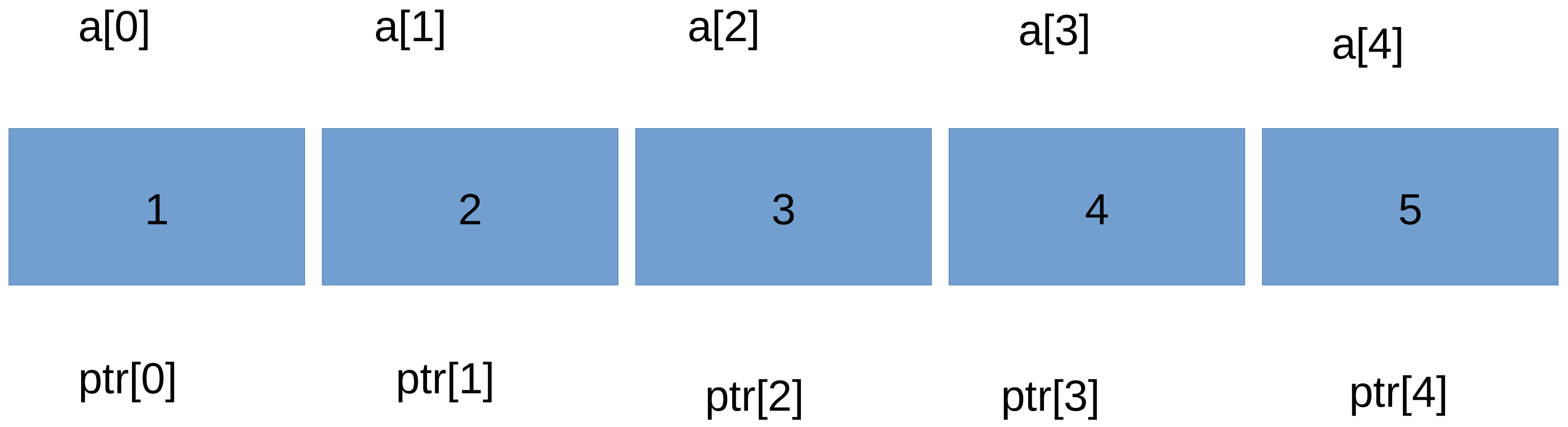
Example

```
#include <stdio.h>
int main() {
   int x[4];
   int i;

   for(i = 0; i < 4; ++i) {
      printf("&x[%d] = %p\n", i, 
&x[i]);
   }

   printf("Address of array x: %p", 
x);

   return 0;
}
```

There is a difference of 4 bytes between two consecutive elements of array x.
because the size of int is 4 bytes (on our compiler).
Notice that, the address of &x[0] and x is the same. It's because the variable name x points to the first element of the array.

```
#include <stdio.h>
int main()
{
    int a[5]={1,2,3,4,5};   //array initialization
    int *ptr;     //pointer declaration
              /*the ptr points to the first element of the array*/

    ptr=a; /*We can also type simply ptr=&a[0] */

    printf("Printing the array elements using pointer\n");
    for(int i=0;i<5;i++)    //loop for traversing array elements
    {
        printf("\n%x",*ptr);  //printing array elements
        ptr++;    //incrementing to the next element, you can also write   ptr=ptr+1
    }
    return 0;
}
```

Contd...



- Ptr++

Now if this ptr is sent to a function as an argument, then the array val can be accessed in a similar fashion.

```
void arr() {
    // Declare an array
    int val[3] = { 5, 10, 15};
    // Declare pointer variable
    int *ptr;
    // Assign address of val[0] to ptr.
    // We can use ptr=&val[0];(both are same)
    ptr = val ;
    cout << "Elements of the array are: ";
    cout << ptr[0] << " " << ptr[1] << " " << ptr[2];
    return 0;
}
int main() {
    arr();
    return 0;
}
```

Example

Here, if ptr points to the first element in the above example then ptr + 3 will point to the fourth element.

For example,

```
int *ptr;
int arr[5];
ptr = arr;

ptr + 1 is equivalent to &arr[1];
ptr + 2 is equivalent to &arr[2];
ptr + 3 is equivalent to &arr[3];
ptr + 4 is equivalent to &arr[4];
```

POINTERS AND MULTIDIMENSIONAL ARRAYS

- multidimensional array can also be represented with an pointer notation

- two-dimensional array, for example, is actually a collection of one -dimensional arrays

- we can define a two-dimensional array as a pointer to a group of contiguous one- dimensional arrays

Contd…

A two-dimensional array declaration can be written as:

data- type (*ptvar) [expression 2] ;

rather than

data- type array[expression 1] [expression 2];

Contd...

“x” is a two-dimensional integer array having 10 rows and 20 columns

```
int (*x)[20];
```

rather than

```
int x[10][20];
```

“x” is defined to be a pointer to a group of contiguous, one-dimensional, 20-element integer arrays

Contd...

- The item in row 3, column 6 can be accessed by writing either

  a[2][5]

- or

  * (* (x + 2) + 5)

matrix addition using pointers

```
*(*(c+row)+col)= *(*(a+row)+col)+*(*(b+row)+col)
```

Contd…

```
#include<stdio.h>
#define MAXROWS 20

void readinput(int *a[MAXROWS], int nrows, int ncols);

void computesums(int *a[MAXROWS],int *b[MAXROWS],int
*c[MAXROWS],int nrows,int ncols);

void writeoutput(int *c[MAXROWS],int nrows,int ncols);
```

```
main()
{
int row,nrows,ncols;
int *a[MAXROWS],*b[MAXROWS],*c[MAXROWS];
printf("How many rows?");
scanf("%d",&nrows);
printf("\n How many columns");
scanf("%d",&ncols);
for(row=0;row<nrows;++row){
a[row]=(int *)malloc(ncols*sizeof(int));
b[row]=(int *)malloc(ncols*sizeof(int));
c[row]=(int *)malloc(ncols*sizeof(int));
}
printf("\n\nFirst table:\n");
readinput(a,nrows,ncols);
printf("\n\nSecond table:\n");
readinput(b,nrows,ncols);
computesums(a,b,c,nrows,ncols);
printf("\n\nSums of the elements:\n\n");
writeoutput(c,nrows,ncols);
}
```

Contd...

```
void readinput(int *a[MAXROWS],int m,int n) {
    int row, col;
    for(row=0;row<m;++row) {
        for(col=0;col<n;++col) {
            scanf("%d",(*(a+row)+col));
        }
    }
}
```

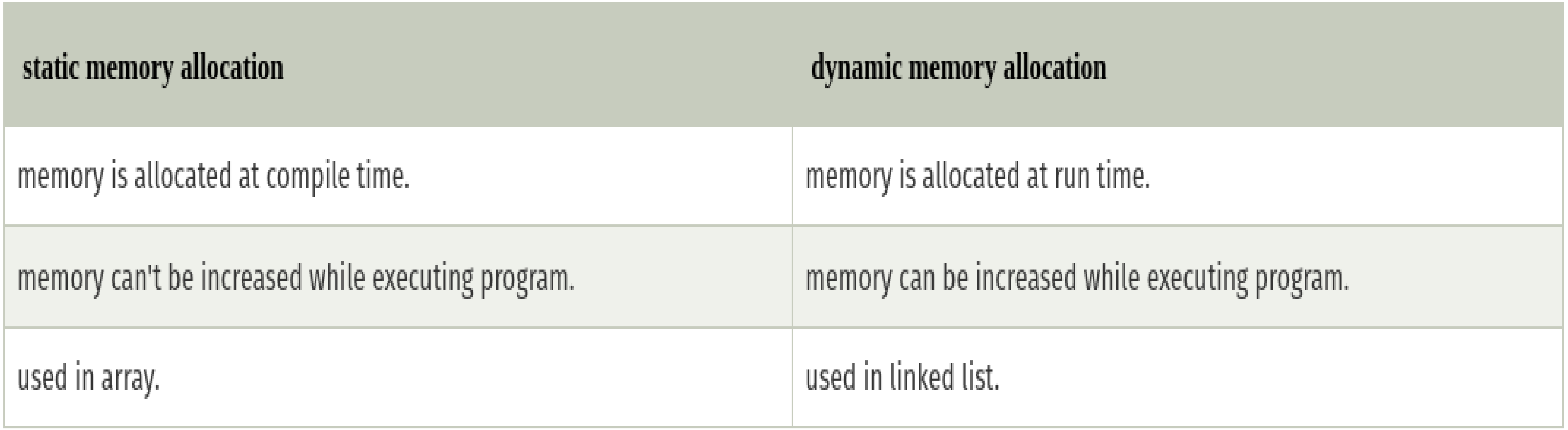
Contd...

```
void computesums(int *a[MAXROWS],int *b[MAXROWS],int *c[MAXROWS],int m,int n)
{
int row, col;
for(row=0;row<m;++row){
for(col=0;col<n;++col){
*(*(c+row)+col) = *(*(a+row)+col)+*(*(b+row)+col);
}
}
}
```

Contd…

```
void writeoutput(int *a[MAXROWS],int m,int n)
{
    int row, col;
    for(row=0;row<m;++row) {
        for(col=0;col<n;++col) {
            printf("%d \t",*(*(a+row)+col));
        }
        printf("\n");
    }
}
```

Static memory allocation and Dynamic memory allocation



static memory allocation	dynamic memory allocation
memory is allocated at compile time.	memory is allocated at run time.
memory can't be increased while executing program.	memory can be increased while executing program.
used in array.	used in linked list.

Dynamic memory allocation

- As we 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.

- To allocate memory dynamically, library functions are malloc(), calloc(), realloc() and free() are used.

- These functions are defined in the <stdlib.h> header file.

Malloc()

- The name "malloc" stands for memory allocation.
- The malloc() function reserves a block of memory of the specified number of bytes.
- It returns a pointer of void which can be casted into pointers of any form.

Syntax for malloc()

```
ptr = (castType*) malloc(size);
```

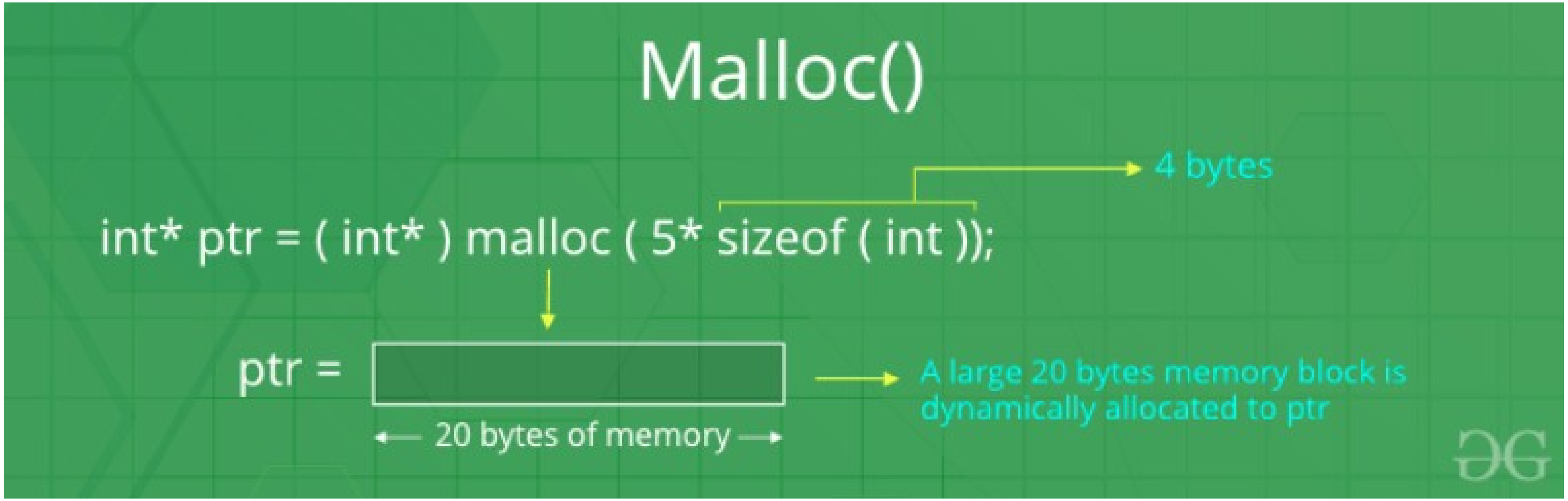
Example:

```
ptr = (float*) malloc(100 * sizeof(float));
```

The above statement allocates 400 bytes of memory. It's because the size of float is 4 bytes. And, the pointer ptr holds the address of the first byte in the allocated memory.

The expression results in a NULL pointer if the memory cannot be allocated.

Contd…


Malloc()
4 bytes
int* ptr = (int*) malloc (5* sizeof (int));
ptr =
20 bytes of memory
A large 20 bytes memory block is dynamically allocated to ptr

Calloc()

The name "calloc" stands for contiguous allocation.

The malloc() function allocates memory and leaves the memory uninitialized,

whereas the calloc() function allocates memory and initializes all bits to zero.

Syntax of calloc()

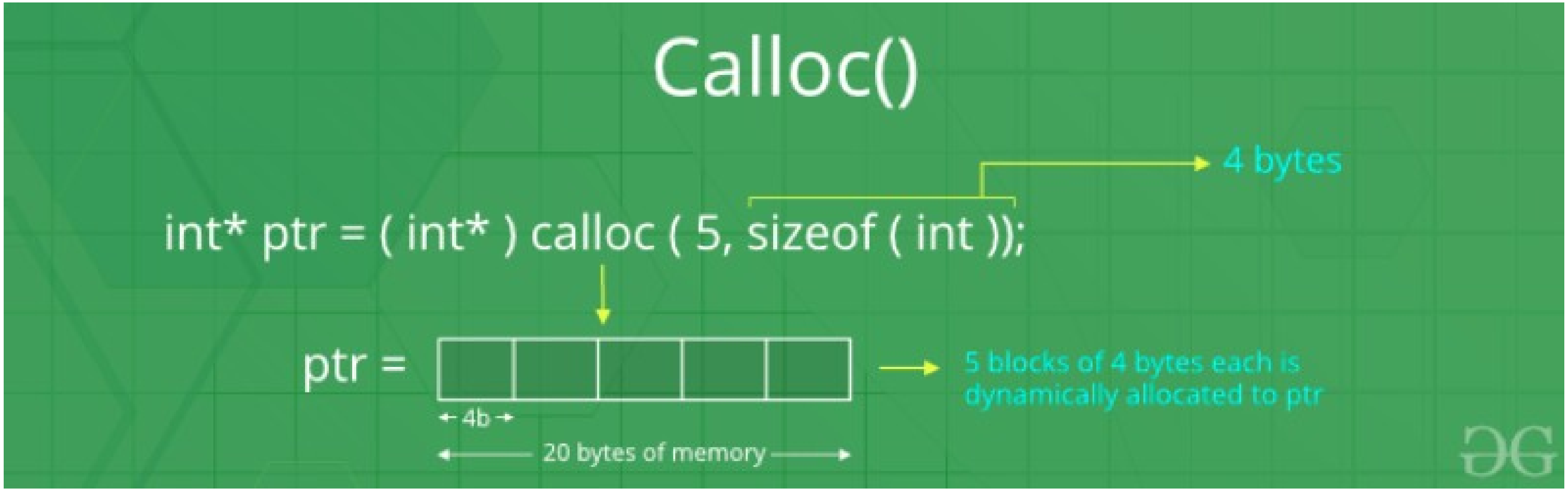
```
ptr = (castType*)calloc(n, size);
```

Example:

```
ptr = (float*) calloc(25, sizeof(float));
```

The above statement allocates contiguous space in memory for 25 elements of type float.

Contd…



Calloc()
4 bytes
int* ptr = (int*) calloc (5, sizeof (int));
ptr =
5 blocks of 4 bytes each is dynamically allocated to ptr
4b
20 bytes of memory

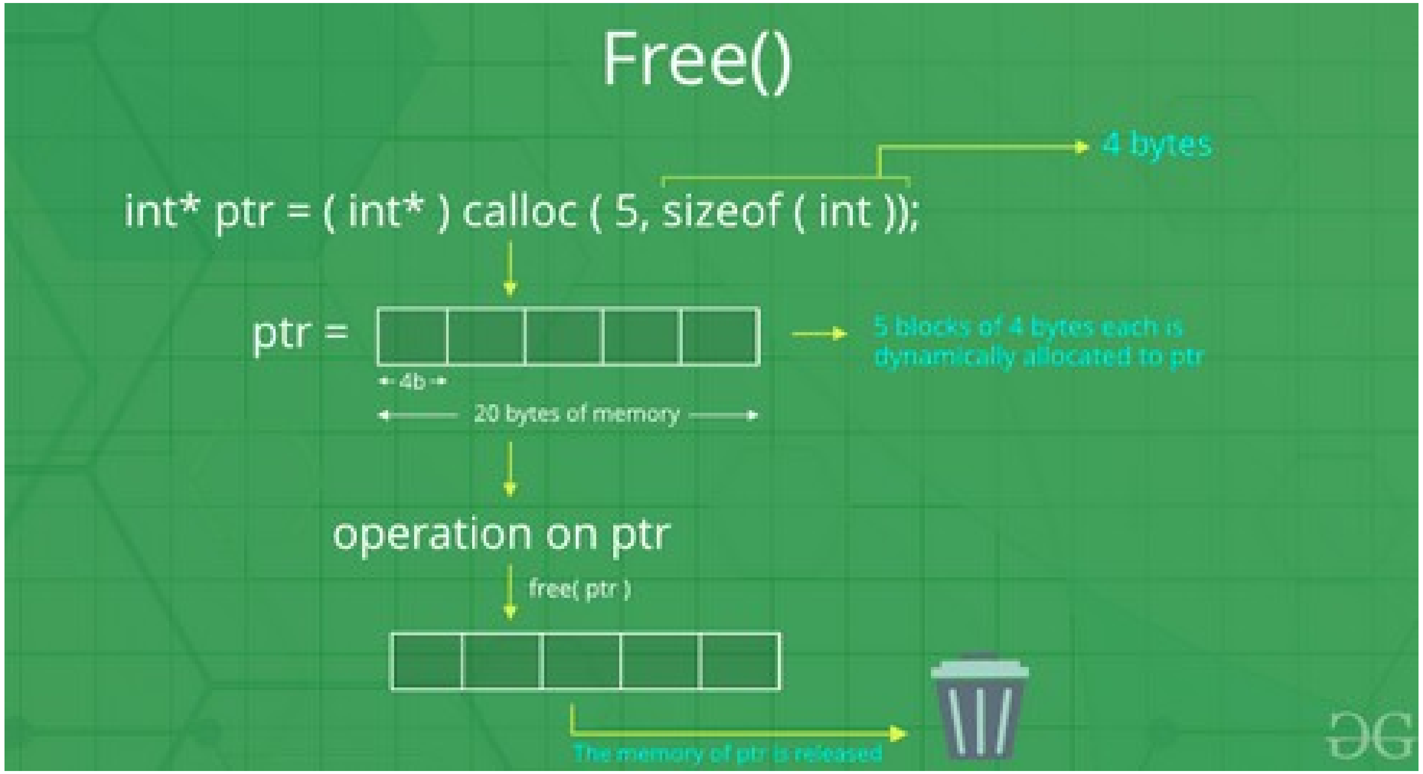
free()

- Dynamically allocated memory created with either calloc() or malloc() doesn't get freed on their own.
- You must explicitly use free() to release the space.
- **Syntax of free()**

  free(ptr);

- This statement frees the space allocated in the memory pointed by ptr.

Contd…


Free()
4 bytes
int* ptr = (int*) calloc (5, sizeof (int));
ptr =
5 blocks of 4 bytes each is dynamically allocated to ptr
4b
20 bytes of memory
operation on ptr
free(ptr)
The memory of ptr is released

Example malloc() and free()

```c
// Program to calculate the sum of n numbers entered by the user
#include <stdio.h>
#include <stdlib.h>
int main() {
  int n, i, *ptr, sum = 0;
  printf("Enter number of elements: ");
  scanf("%d", &n);
  ptr = (int*) malloc(n * sizeof(int));

  // if memory cannot be allocated
  if(ptr == NULL) {
    printf("Error! memory not allocated.");
    exit(0);
  }
  printf("Enter elements: ");
  for(i = 0; i < n; ++i) {
    scanf("%d", ptr + i);
    sum += *(ptr + i);
  }
  printf("Sum = %d", sum);

  // deallocating the memory
  free(ptr);
  return 0;
}
```

```
Enter number of elements: 3
Enter elements: 100
20
36
Sum = 156
```

Example calloc() and free()

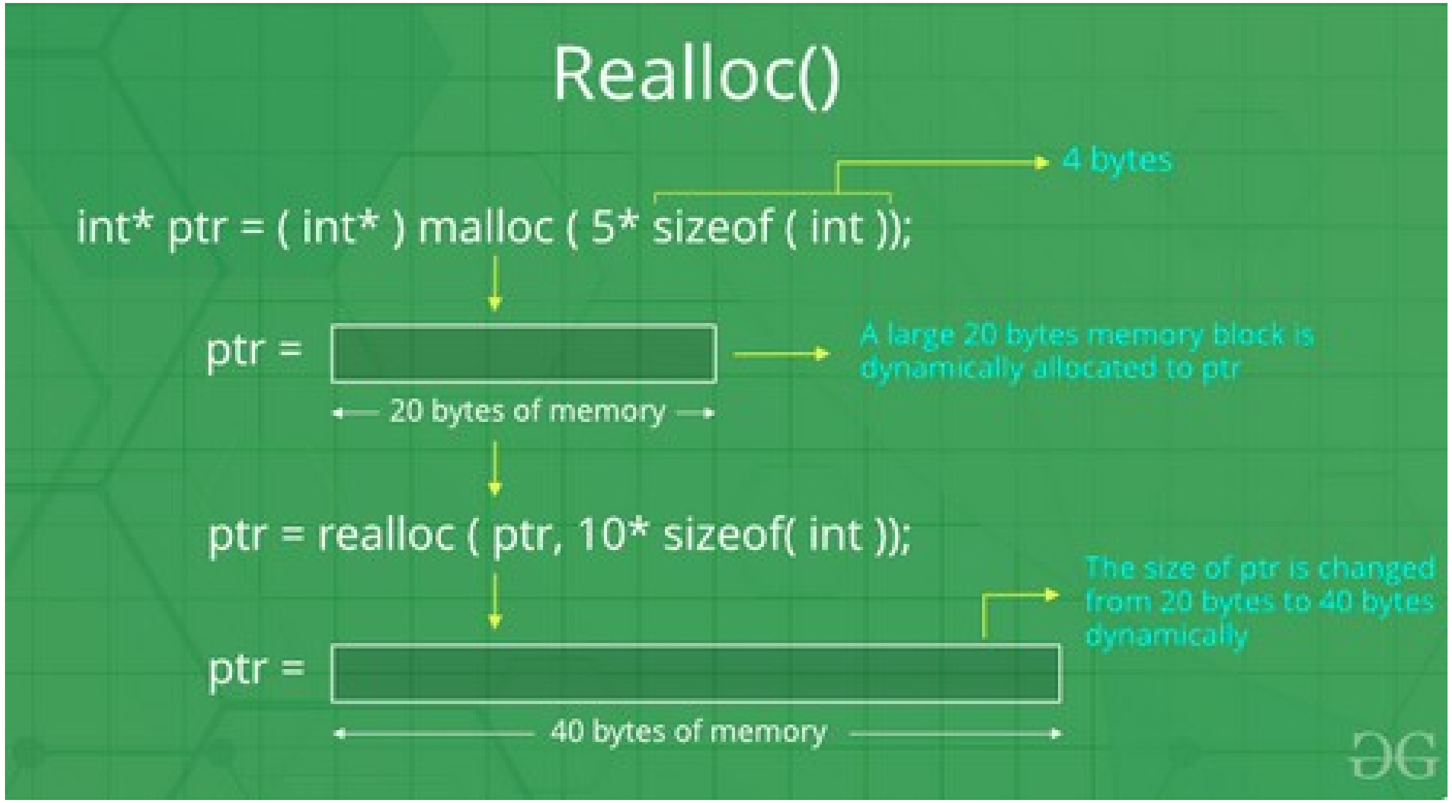
```
// Program to calculate the sum of n numbers entered by the user
#include <stdio.h>
#include <stdlib.h>

int main() {
  int n, i, *ptr, sum = 0;
  printf("Enter number of elements: ");
  scanf("%d", &n);

  ptr = (int*) calloc(n, sizeof(int));
  if(ptr == NULL) {
    printf("Error! memory not allocated.");
    exit(0);
  }

  printf("Enter elements: ");
  for(i = 0; i < n; ++i) {
    scanf("%d", ptr + i);
    sum += *(ptr + i);
  }

  printf("Sum = %d", sum);
  free(ptr);
  return 0;
}
```

```
Enter number of elements: 3
Enter elements: 100
20
36
Sum = 156
```

realloc()

- If the dynamically allocated memory is insufficient or more than required, you can change the size of previously allocated memory using the realloc() function.
- **Syntax of realloc()**
- ptr = realloc(ptr, x);
- Here, ptr is reallocated with a new size x.

Contd…


Realloc()
4 bytes
int* ptr = (int*) malloc (5* sizeof (int));
ptr =
A large 20 bytes memory block is dynamically allocated to ptr
20 bytes of memory
ptr = realloc (ptr, 10* sizeof(int));
The size of ptr is changed from 20 bytes to 40 bytes dynamically
ptr =
40 bytes of memory

Example realloc() and free()

```
#include <stdlib.h>

 int *ptr, i , n1, n2;
 printf("Enter size: ");
 scanf("%d", &n1);

 ptr = (int*) malloc(n1 * sizeof(int));

 printf("Addresses of previously allocated memory:\n");
 for(i = 0; i < n1; ++i)
   printf("%pc\n",ptr + i);

 printf("\nEnter the new size: ");
 scanf("%d", &n2);

 // rellocating the memory
 ptr = realloc(ptr, n2 * sizeof(int));

 printf("Addresses of newly allocated memory:\n");
 for(i = 0; i < n2; ++i)
   printf("%pc\n", ptr + i);

 free(ptr);
```

```
Enter size: 2
Addresses of previously
allocated memory:
26855472
26855476

Enter the new size: 4
Addresses of newly
allocated memory:
26855472
26855476
26855480
26855484
```

More Example

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int index = 0, i = 0, n,
        *marks; // this marks pointer hold the base address
                // of  the block created
    int ans;
    marks = (int*)malloc(sizeof( int)); // dynamically allocate memory using malloc
    // check if the memory is successfully allocated by
    // malloc or not?
    if (marks == NULL) {
        printf("memory cannot be allocated");
    }
    else {
        // memory has successfully allocated
        printf("Memory has been successfully allocated by "  "using malloc\n");
        printf("\n marks = %p\n", marks); // print the base or beginning
                       // address of allocated memory
        do {
            printf("\n Enter Marks\n");
            scanf("%d", &marks[index]); // Get the marks
            printf("would you like to add more(1/0): ");
            scanf("%d", &ans);

            if (ans == 1) {
                index++;
                marks = (int*)realloc( marks, (index + 1)  * sizeof( int)); // Dynamically reallocate
                 // memory by using realloc
                // check if the memory is successfully
                // allocated by realloc or not?
                if (marks == NULL) {
                    printf("memory cannot be allocated");
                }
                else {
                    printf("Memory has been successfully "  "reallocated using realloc:\n");
                    printf("\n base address of marks are:%p", marks); ////print the base or
                                ///beginning address of
                                ///allocated memory
                }
            }
        } while (ans == 1);
        // print the marks of the students
        for (i = 0; i <= index; i++) {
            printf("marks of students %d are: %d\n ", i,
                   marks[i]);
        }
        free(marks);
    }
    return 0;
}
```

Any queries???