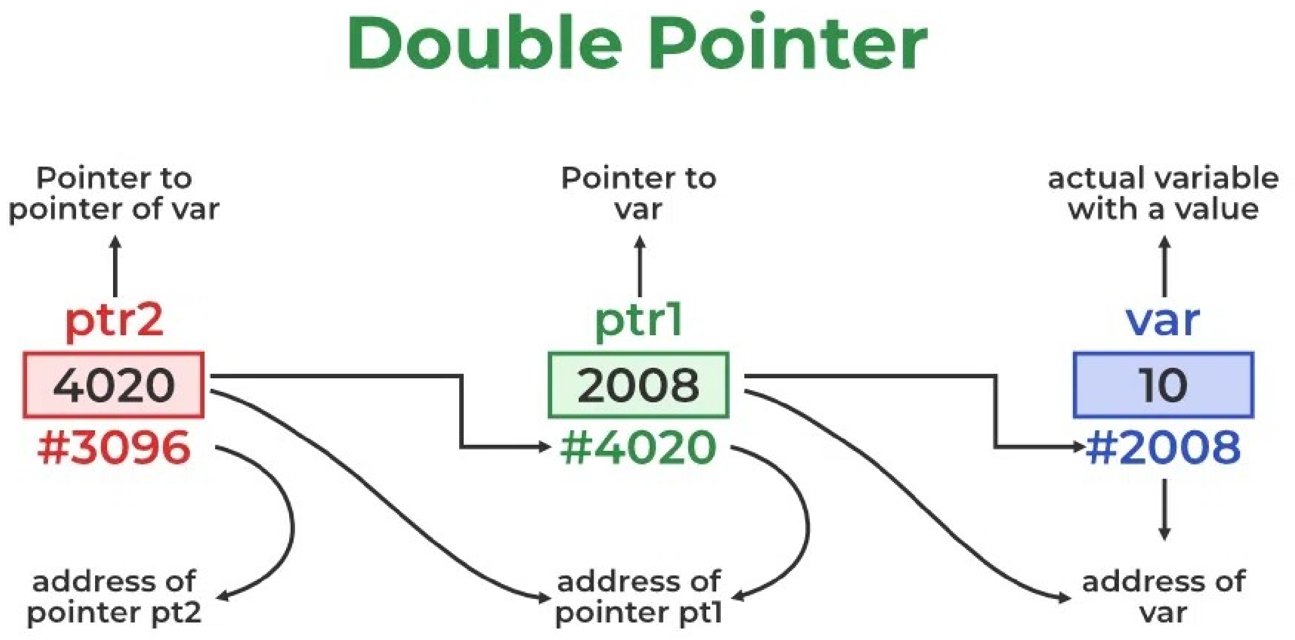
C PROGRAMMING
Lecture 6

1st semester 2023-2024

Pointers to Pointers

- A pointer is a variable that can store the address of another variable. The other variable can also be a pointer; so it means that a pointer can point to another pointer.



https://www.geeksforgeeks.org/c-pointer-to-pointer-double-pointer/

Pointers to Pointers

Definition:

A pointer to a pointer is declared by using two asterisks (**) in front of the variable name.
For example, int **pp; declares a pointer to a pointer to an integer.

Purpose:

Pointers to pointers are used to handle complex data structures or dynamically allocated memory.
They are often used for modifying a pointer from within a function and for working with multi-dimensional arrays.

Pointers to Pointers

Memory Representation:

A pointer to a pointer is a variable that stores the memory address of another pointer. It essentially points to a pointer variable.

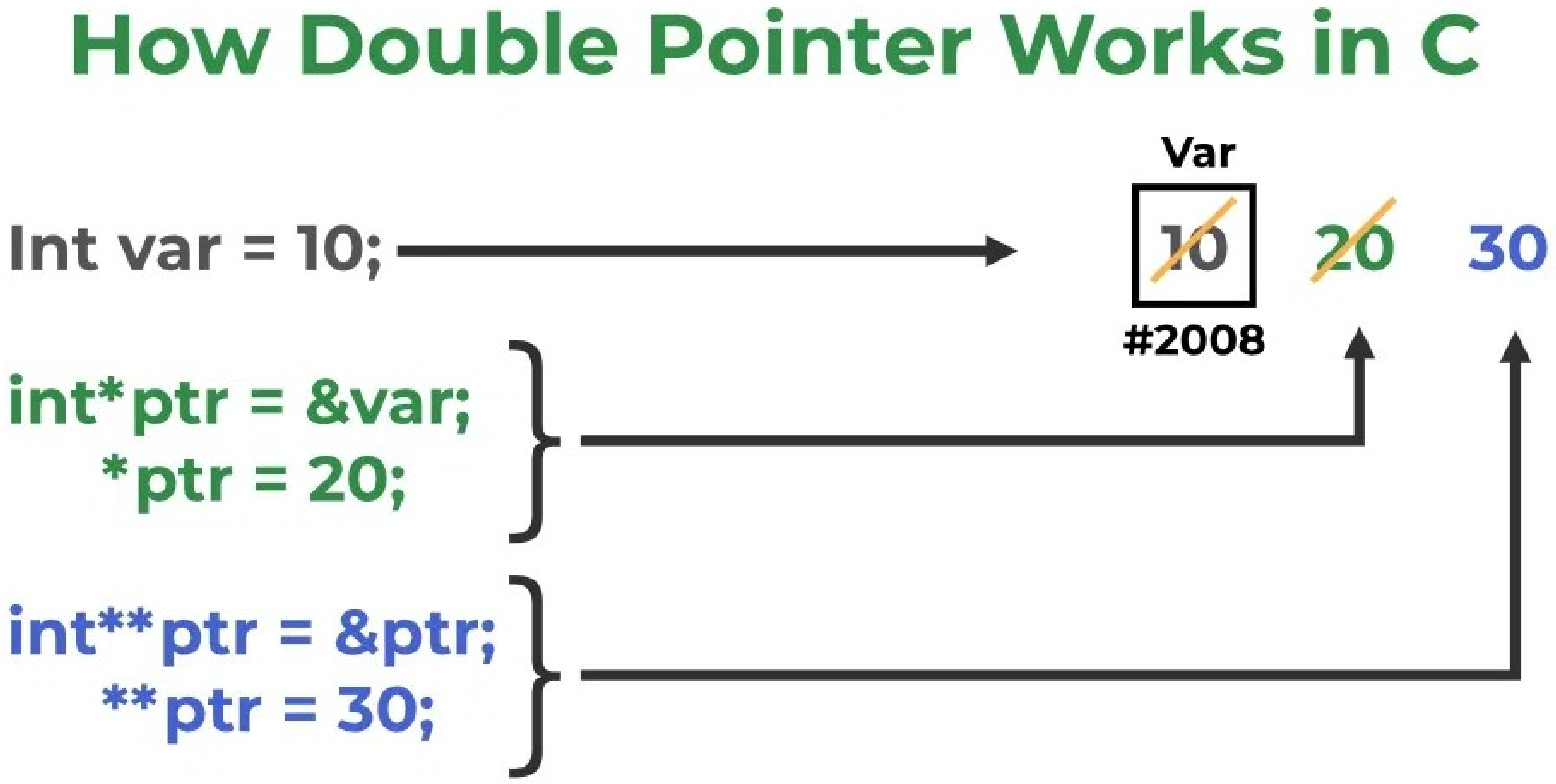
The first pointer (the one being pointed to) typically points to some data.

```
int main() {
    int x = 42;
    int *p = &x; // Pointer to int
    int **pp = &p; // Pointer to pointer to int

    printf("Value of x: %d\n", **pp); // Accessing the value pointed to by the
pointer to pointer

    return 0;
}
```

Pointers to Pointers

```
#include<stdio.h>
int main()
{
    int **ptr1 = NULL;
    int *ptr2 = NULL;
    int nr1 = 0;
    ptr2 = &nr1;
    ptr1 = &ptr2;
    printf("\n nr1 = [%d]\n",nr1);
    printf("\n *ptr2 = [%d]\n",*ptr2);
    printf("\n **ptr1 = [%d]\n",**ptr1);
    return 0;
}
```

Pointers to Pointers



https://www.geeksforgeeks.org/c-pointer-to-pointer-double-pointer/

Pointers to Pointers

Use Cases:

<u>Dynamic Memory Allocation</u>: Pointers to pointers are used when you allocate memory dynamically and need to maintain a reference to the pointer variable.

<u>Multi-Dimensional Arrays</u>: In multi-dimensional arrays, you can use pointers to pointers to create arrays of pointers, making it easier to work with arrays of different sizes.

<u>Function Parameter Modification</u>: They are often used when you need to modify a pointer or allocate memory inside a function and retain those changes outside the function.

Pointer Arrays

- An array of pointers can be declared as :

```
<type> *<name>[<number_of_elements];
```

- For example :

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

declares an array of five pointers to integer numbers.

Pointer Arrays

```
#include<stdio.h>
int main()
{
int nr1=0, nr2=2;
    int *arr[2];    arr[0] = &nr1;    arr[1] = &nr2;
   printf("\n nr1 = [%d] \n",nr1);
   printf("\n nr2 = [%d] \n",nr2);
   printf("\n arr[0] = [%p] \n",arr[0]);
   printf("\n arr[1] = [%p] \n",arr[1]);
   printf("\n val of *arr[0] = [%d] \n",*arr[0]);
   printf("\n val of *arr[1] = [%d] \n",*arr[1]);
   return 0;
}
```

Function Pointers

- Just like pointer to characters, integers etc, we can have pointers to functions
- Declaration:

```
<return_type> (*<pointer_name>) (type_of_
  arguments)
```

- The same as for arrays, the function name is the address of the function
- A pointer to a function can be manipulated in the same way as other pointers; most important, it can be passed to a function

Function Pointers

- When to use function pointers?
  - Passing function as parameters for functions
  - Callback functions
- Declaration example:

```
int (*func)(int)
```

Beware that is different from

```
int *funct(int)
```

```
(*func)(int) is used to declare and work with
  function pointers.
*func(int) is used to declare and define
  functions
```

Function Pointers

- Example:

```
void makesomething(int nr1, int nr2, int (*func)(int))
{
   int i;
   for(i=nr1;i<=nr2;i++)
      printf("%d %d\n", i, (*func)(i));
}
```

- How to interpret the printf from example:

func is a pointer to a function; *func is the function
i is the argument of the function, passed as arg between ()
the value returned is int, matched by %d

Function Pointers

```
#include <stdio.h>
void performtask(int nr1, int nr2, float (*func)
(int))
{
   int i;
   for(i=nr1;i<=nr2;i++)
      printf("%d %f\n", i, (*func)(i));
}
float reciprocal(int nr)
{
   return(1.0/nr);
}
```

Function Pointers

```
float square(int nr)
{
   return(nr*nr);
}
int main()
{
   performtask(1,5,reciprocal);  // can be called
also square, depending on some condition…
  return 0;
}
```