Day 2 (02-12-2025)

Type modifiers & classifiers

Modifiers

- signed : Allows -ve and +ve.
- unsigned : Allows only +ve, -ve values are not allowed.
- short : It is used to short the range of datatype
- long : It is used to make the range higher than normal range

Classifiers

- const : to make variables, functions and classes constant/read-only.
- volatile : to make variable volatile for changes that may occur in future(mostly via hardware or any related means)

Pointers:

The size of pointer is not dependent on datatype, it depends on your system architecture/compiler architecture.

- In 32 bit system, size of pointer is 4 bytes.
- In 64 bit system, size of pointer is 8 bytes.

Pointer Declaration & Assignment

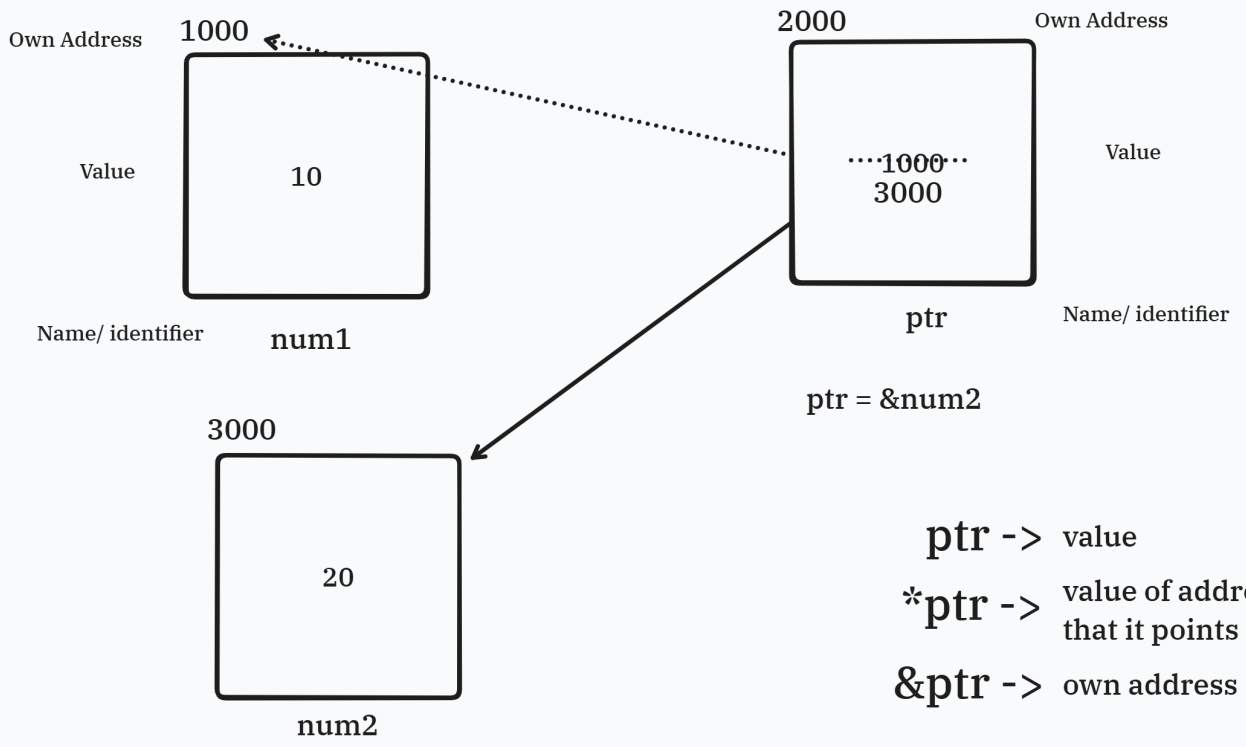
```
//pointers ka basics
int main(){

    int num1 = 10; //initiliazation of variable

    int *ptrNum1 = &num1;  // initialization of pointer variable

    int num2 = 20; //2000

    printf("Num1 Value: %d \n", num1);  // num ka value print
    printf("ptrNum1 Value via ptr: %p \n", ptrNum1); // ptrnNum1 ka value print
i.e. address of another variable
    printf("Num1 value via ptr : %d \n", *ptrNum1);  // value of num1 via
dereferencing from ptr

//  *ptrNum1 = &num2;  // not valid

    ptrNum1 = &num2; // reassign the address of num2

    printf("Num2 Value: %d \n", num2);  // num ka value print
    printf("ptrNum1 value via ptr: %p \n", ptrNum1);
    printf("Num2 value via ptr : %d \n", *ptrNum1);
```

```
    *ptrNum1 = 30; // changing the value of variable via dereferencing

    printf("Num2 Value: %d \n", num2);

    return 0;
}
```

int num1 = 10;

int *ptr = &num1; -> Declaring/initialization a variable
which is a pointer that stores a address

*ptr -> Dereferencing: to get the value of address it is points to

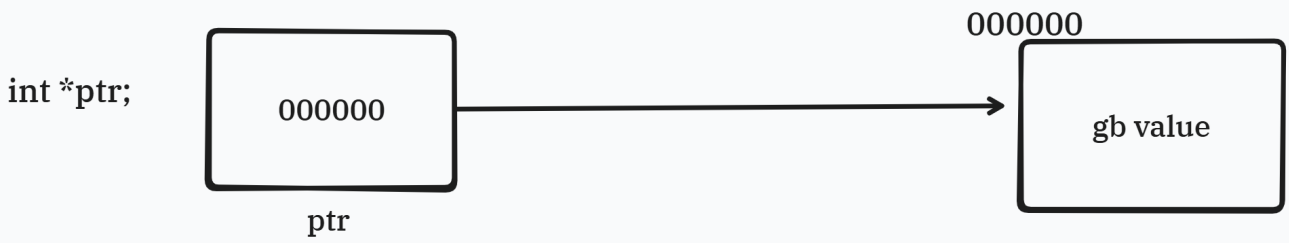


ptr -> value
*ptr -> value of address that it points to
&ptr -> own address

Wild pointer

- A pointer that is not initialized to any address is called a wild pointer.
- It can point to any random memory location.

```
int main(){

    int *ptrNum1; // wild pointer

    printf("ptrNum1 Value: %p \n", ptrNum1); // random address
    //printf("Num1 value via ptr : %d \n", *ptrNum1); // not valid, it can cause
segmentation fault
}
```



NULL & nullptr

- NULL is a macro defined in stdio.h and it is used to represent a null pointer.
- nullptr is a keyword introduced in C++11 and it is used to represent a null pointer
- In C, we use NULL to initialize a pointer to a null value.
- In C++, we use nullptr to initialize a pointer to a null value.

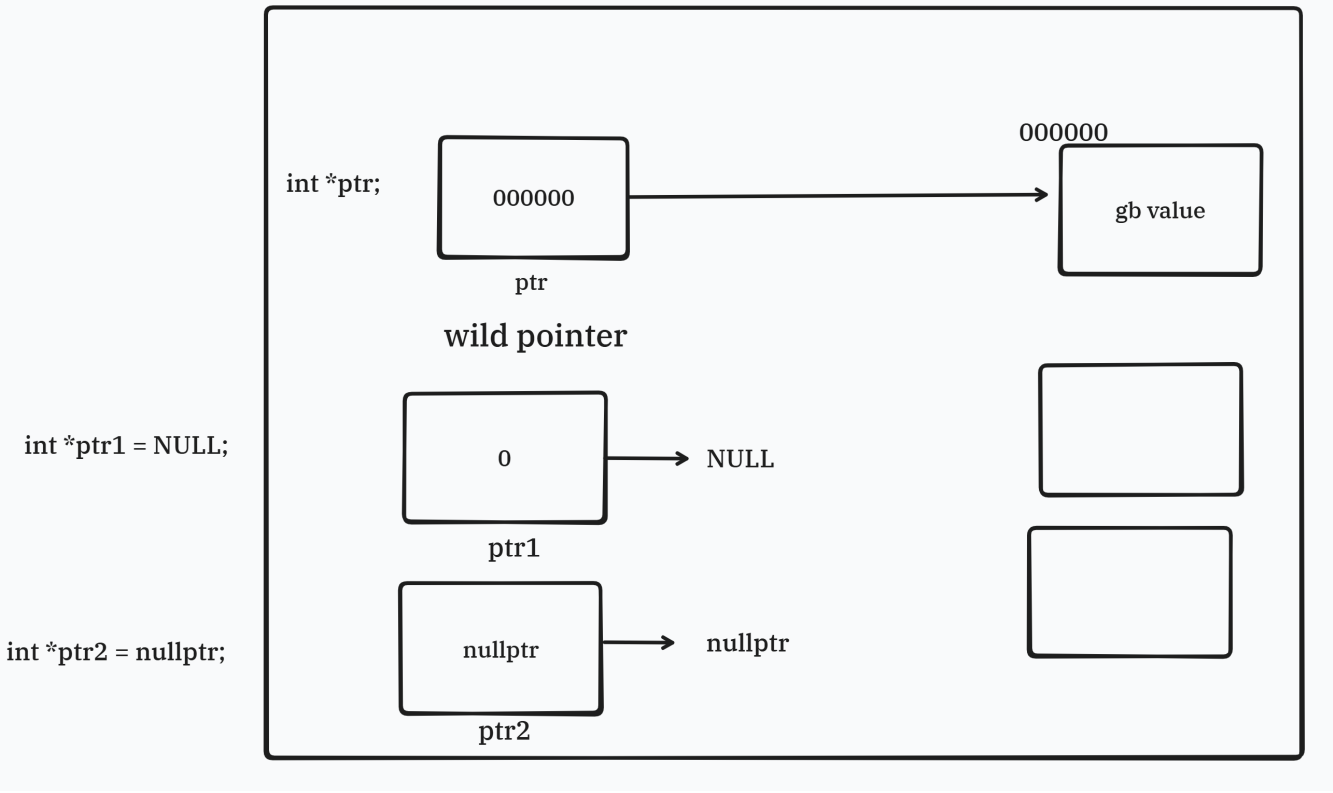
```
int main(){

    int *ptrNum1 = NULL; // NULL pointer

    if(ptrNum1 == NULL){
        printf("ptrNum1 is a NULL pointer \n");
    }

    ptrNum1 = nullptr; // not valid in C, valid in C++
    if(ptrNum1 == nullptr){
        printf("ptrNum1 is a nullptr \n");
    }

    return 0;
}
```

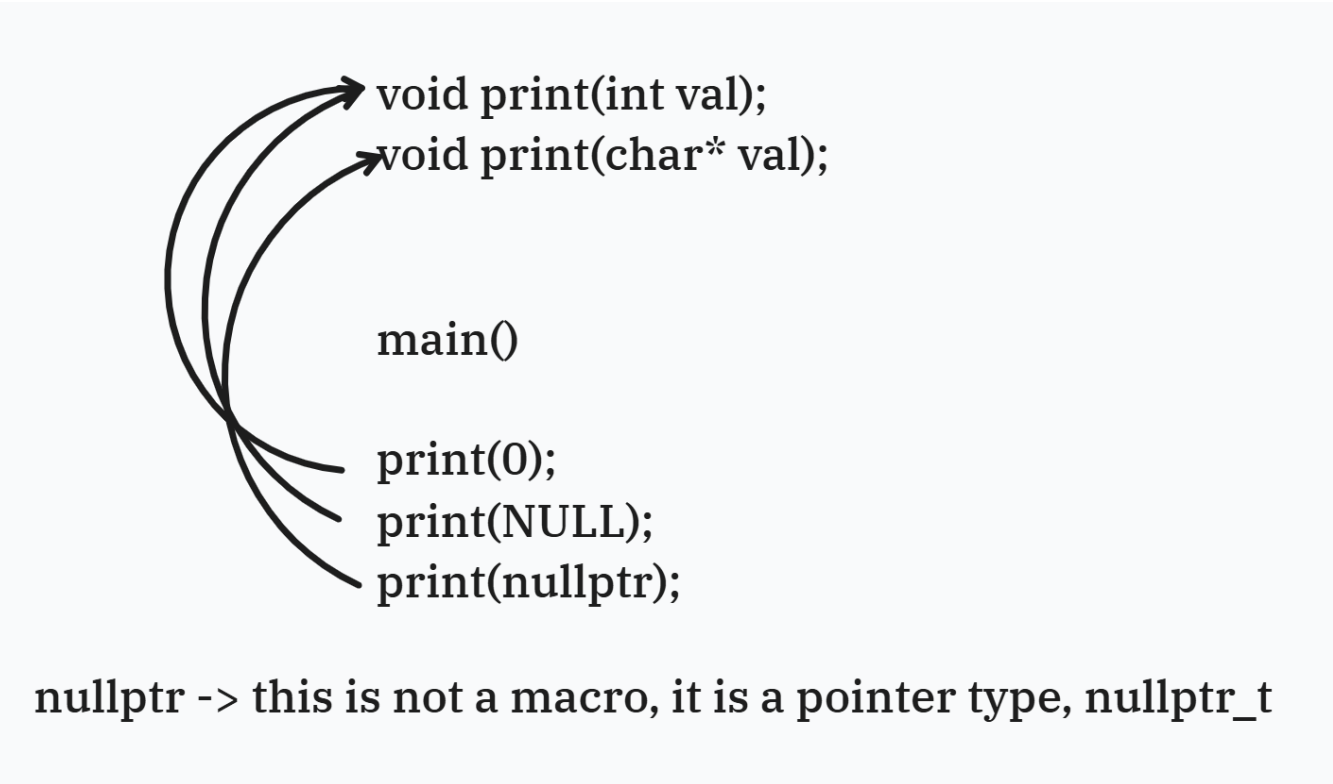
Function overloading with pointers

- With function overloading, NULL can lead to unexpected behavior because NULL is defined as 0, which can match both integer and pointer types.
- To avoid this, use `nullptr` in C++ which is a type-safe null pointer

```
void func(int i) {
    printf("Calling func(int)\n");
}

void func(char* p) {
    printf("Calling func(char*)\n");
}

int main() {
```

```
    func(0);        // Output: Calling func(int) (Correct)
    func(NULL);     // Output: Calling func(int) (Unexpected, NULL is 0)
    func(nullptr);  // Output: Calling func(char*) (Correct, nullptr is pointer-
type)
    return 0;
}
```



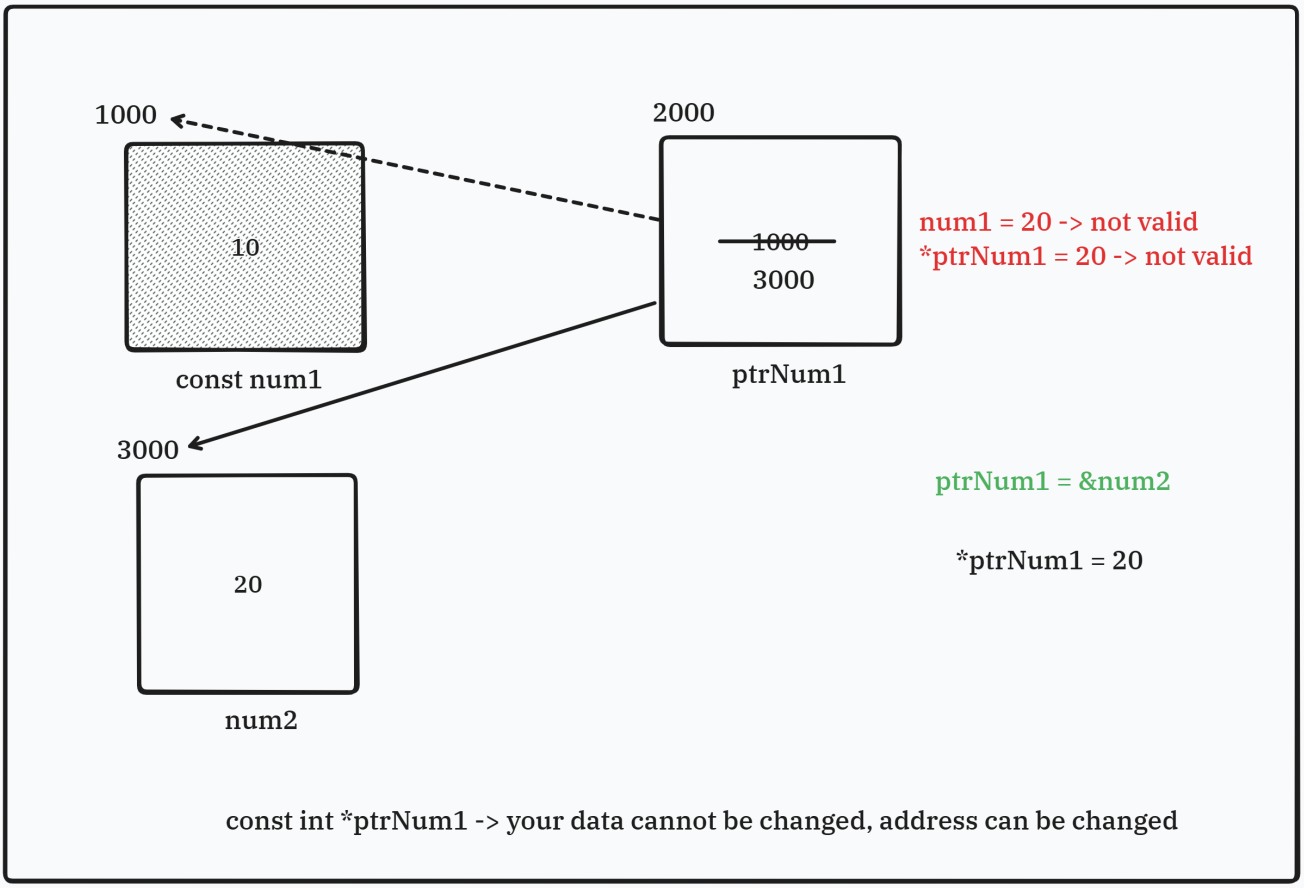


POINTER + CONST combinations

Examples:

const int *ptr & int const *ptr:

```
const int num1 = 10; // constant local variable

//  int *ptrNum1 = &num1; // pointer to num1  //not valid

    const int *ptrNum1 = &num1; // data cannot be changed, but can point to new
address.

    int num2 = 20;

    ptrNum1 = &num2; // valid

//  *ptrNum1 = 20; // not valid

    printf("%d \n", *ptrNum1);

    return 0;
```

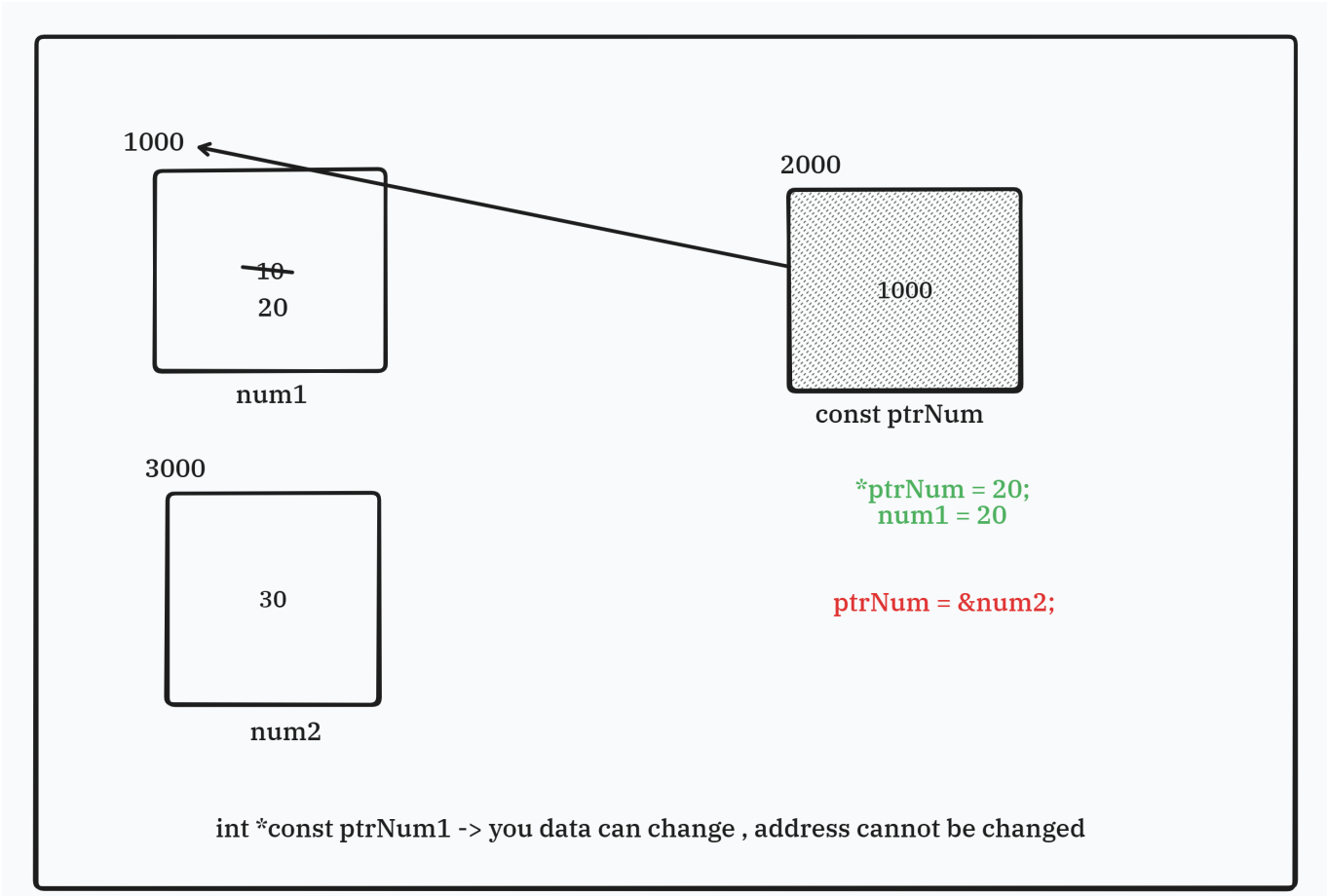



```
const int num1 = 10; // constant local variable

    //Both are identical
```

```
    int const *ptrNum1 = &num1;  // constant ptr variable
//  const int *ptrNum1 = &num1;

    int num2 = 20;

    ptrNum1 = &num2; //

//  *ptrNum1 = 20; // data cannnot be changed //error: assignment of read-only
location '* ptrNum1'

    printf("%d \n", *ptrNum1);

    return 0;
```

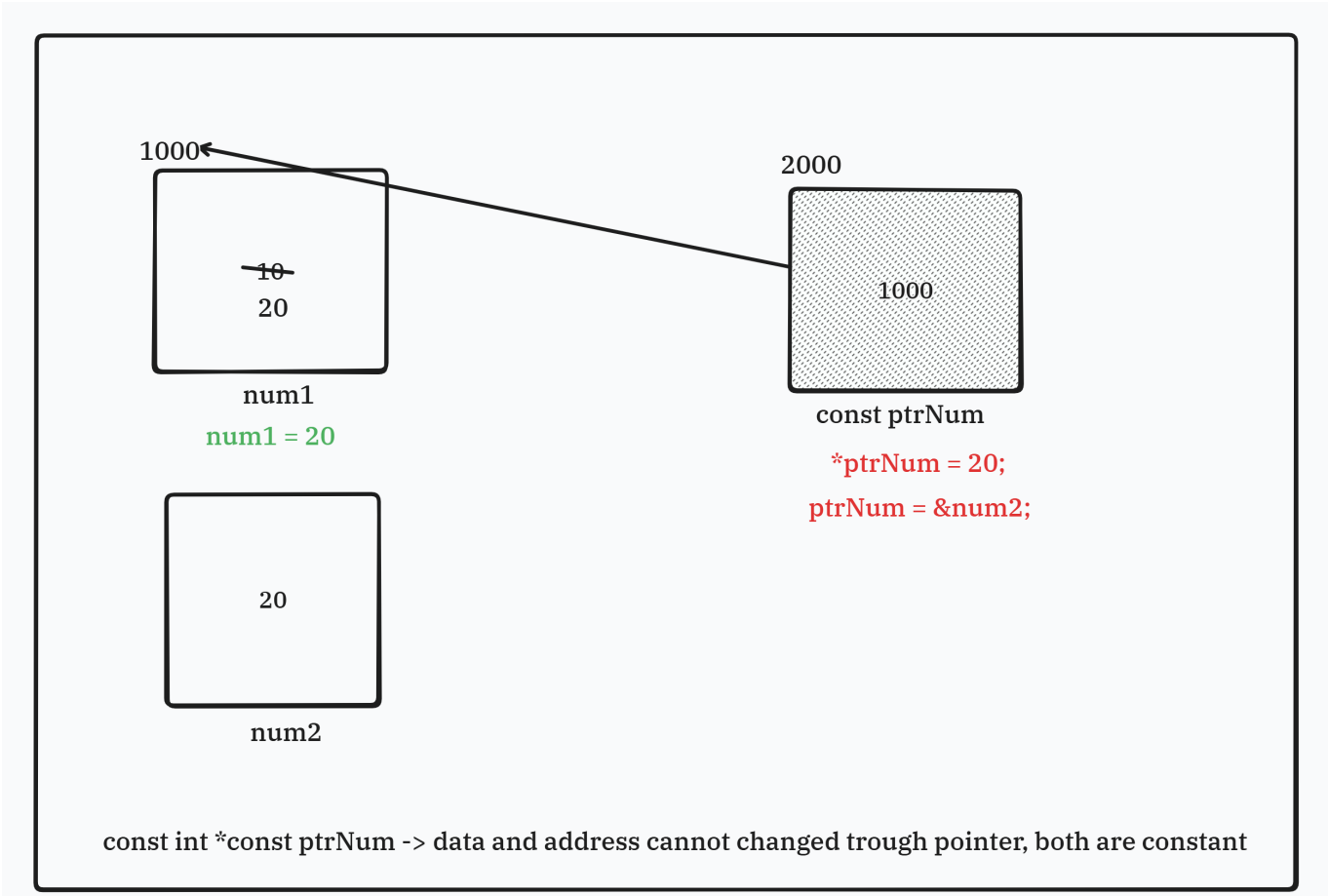
int *const ptr:

```
int num1 = 10; //non constant variable

    int *const ptrNum = &num1;

    int num2 = 30;  // non constant variable

    *ptrNum = 20;  // valid

//  ptrNum = &num2;  // ptrNum -> cannot point to another address -> it is
constant

    printf("%d \n", *ptrNum);

    return 0;
```



const int *const ptr & int const *const ptr:

```c
int num1 = 10;  //non constant variable

    //Both are same and valid
//  int const *const ptrNum = &num1;  // valid
    const int *const ptrNum = &num1;  // const pointer variable with const data
access

//  *ptrNum = 20;  //not valid

    int num2 = 20;

//  ptrNum = &num2;   // not valid
    num1 = 20;  // normal assignment for non constant variable

    printf("%d \n", *ptrNum);

    return 0;
```



const int const *ptr:

```c
int num1 = 10;

const int const *ptrNum = &num1; //not valid -> duplicate const

    printf("%d \n", *ptrNum);

    return 0;
```

Structure in C

- A structure is a user-defined data type in C that allows us to combine data of different types.
- It is used to group related variables together.
- It is defined using the `struct` keyword.
- By default public -> from any place/line in the code we can access the values directly via object
- We cannot have member function, constructor, destructor in the struct in c

```c
#include <stdio.h>

struct Student{
    int rollNo;
    char name[20];
```

```c
};

//global function
void setData(struct Student *s1){ // pointer
    printf("Enter Student Roll No: \n");
    fflush(stdout);
    scanf("%d", &s1->rollNo);
    printf("Enter Student Name: \n");
    fflush(stdout);
    scanf("%s", s1->name);
}
// global function
void printData(struct Student *s1){
    printf("===Student Detail=== \n");
    printf("Roll No. : %d \n", s1->rollNo);
    printf("Name     : %s \n", s1->name);
}

int main(){

    struct Student s1; // In c it is mandatory to write struct

    //  setData(s1);  // s1 -> kuch to data set kiya
    //  printData(s1); // s1 -> this will be another object

    setData(&s1); //address pass to setData(&)
    printData(&s1); // address pass to printData(&)

    return 0;
}
```

- Tommorow:
  - C++ Intro
  - Structure in C++ -> typedef
  - Access specifiers
  - Class & Object