Pointers in C

Course Title :- Structured Programming Language Sessional
Course Code :- CSE-122 [SECTION-B]
Level Term: 1-II-A(G1) & 1-II-B(G3,G4)

Khandaker Jannatul Ritu, Lecturer(CSE), BAIUST

Outlines

1. How data is stored in memory?
2. Introduction to pointer
3. Summery of pointers basics
4. Pointer pointing to different variables
5. Sub and sub of two pointer
6. Practice questions: 1, 2, 3, 4
7. C – Program to illustrate Call-By-Value and Call-By-Reference
8. C – Program to Swap Two variable using pointer and function
9. C – Program to access array elements using pointers
10. What is a Pointer in C? Syntax of C Pointers
11. How to Use Pointers?
12. Various Types of Pointers in C
13. C Pointer Arithmetic
14. Uses of Pointers in C
15. What are the differences between an array and a pointer?
16. Advantage of pointer
17. Relationship between Arrays and Pointers
18. Exercise problems

❑ How data is stored in memory?

Example -1 :

মনে করো একটা ভ্যারিয়েবল নিলাম x এবং
x এর মধ্যে ডাটা রাখলাম 5

```
int main()
{
    int x = 5;
}
```

প্রশ্ন-১ :- x এর মধ্যে না হয় 5 রাখলাম, কিন্তু 5 কম্পিউটারে কোন ফ্যারমেটে স্টোর হয়?
উত্তর :- বাইনারি ডাটা ফ্যারমেট হিসাবে, অর্থাৎ 5 স্টোর হবে 00000101 [৮- বিট রেজিস্টার] হিসাবে মেমোরিতে স্টোর হবে।

Value of X	00000101
Memory address of X	?

প্রশ্ন-২ :- যেহেতু x = 5, সেহেতু x এর মধ্যে ৫ কে রাখা হয়েছে। ৫ নাহয় x এর মধ্য রাখা কিন্তু ভ্যারিয়েবল x কে কম্পিউটারে কিভাবে স্টোর করা হয়?
উত্তর :- কম্পিউটারে তো ০ আর ১ ছাড়া কিছু চিনে না, তাহলে এমন তো সম্ভব নয় যে ডাইরেক্ট x কে রেখে দিলাম।

Value of X	00000101
Memory address of X	X

তাহলে আমাদের প্রধান প্রশ্ন হচ্ছে x এর মধ্যে ৫ রাখা আছে, কিন্তু x মেমোরির কোথায় রাখা আছে?

Value of X	5
Memory address of X	?

Example - 2 :

```
int main()
{
    int x = 5, y = 6, z = 7;
}
```

আরও কিছু ভ্যারিয়েবল নেই x, y, z

এরা কিভাবে **স্টোর** হবে?

Memory Location of Variable	Value
x	5
y	6
z	7

Memory Location of Variable	Value
x	101
y	110
z	111

Memory Location of Variable	Value
?	101
?	110
?	111

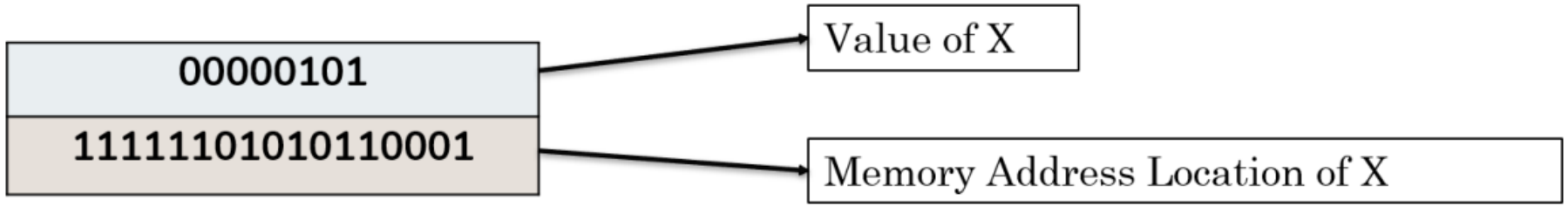
১. ভ্যারিয়েবল এর মধ্যে রাখা ভ্যালু বাইনারি ফরমেটে স্টোর হবে
২. কিন্তু ভ্যারিয়েবল নিজে কিভাবে স্টোর হবে?

কম্পিউটারে তো ০ আর ১ ছাড়া কিছু চিনে না, তাহলে এমন তো সম্ভব নয় যে ডাইরেক্ট x = 5, y = 6, z = 7 কে রেখে দিলাম।

তাহলে আমাদের প্রধান প্রশ্ন হচ্ছে X / Y / Z মেমোরির কোথায় রাখা আছে? কোন ফরমেটে রাখা আছে?

Solution of Ex- 1 & 2

সহজ উত্তর, ভ্যারিয়েবলকেও বাইনারি ফরমেটেই রাখা হয়ে থাকে। কারন কম্পিউটার ৫,৬,৭ ও চিনে না, আর x,y,z ও চিনেনা। কম্পিউটার শুধু ০ আর ১ চিনে।
আমরা যখন লিখি x = 5. তখন মেমোরিতে স্টোর হবে এই ভাবেঃ-



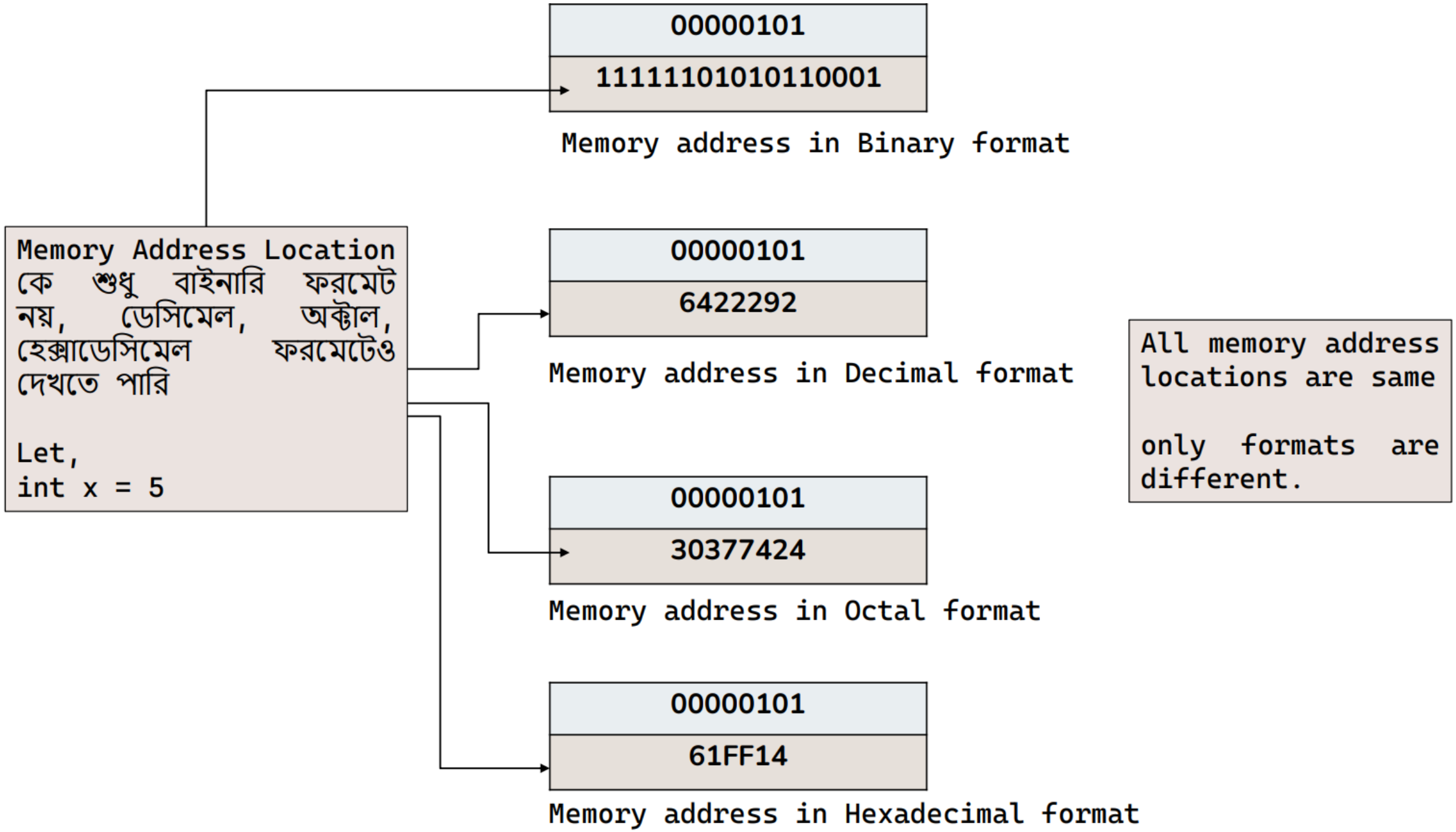
Q-1. এখন যদি বলি 5 [00000101] কোথায় রাখা আছে?
➔ X এর মধ্যে

Q-2. X কই রাখা আছে?
➔ 1111110101010110001 **এই লোকেশনে**

Memory Location of Variable	Value
x	101
y	110
z	111

Memory Location of Variable	Value
1111110101010110001	101
1100101010110001	110
0001101010110001	111

N.B.:- মূল কথা,
মেমোরিতে 5 এর যেমন corresponding binary value store হয়
ঠিক একিই ভাবে X corresponding binary value store হয় এবং একে বাইনারি লোকেশনকে Memory Address Location of X বলা হয়


00000101
1111110101010001
Memory address in Binary format
Memory Address Location কে শুধু বাইনারি ফরমেট নয়, ডেসিমেল, অক্টাল, হেক্সাডেসিমেল ফরমেটেও দেখতে পারি
Let,
int x = 5
00000101
6422292
Memory address in Decimal format
All memory address locations are same
only formats are different.
00000101
30377424
Memory address in Octal format
00000101
61FF14
Memory address in Hexadecimal format

এবার আসি নেক্সট ধাপে,
Let,
যদি একটা ভ্যারিয়েবল int x = 5 লিখি,

১. X এর ভ্যালু কিভাবে প্রিন্ট করতে হবে?

এইভাবে printf(“ %d ”, x); প্রিন্ট করি। তাহলে X এর ভ্যালু প্রিন্ট হবে, অর্থাৎ 5 প্রিন্ট হবে। যা খুবই সহজ কাজ।

২. কিন্তু X এর Memory address location কিভাবে প্রিন্ট করতে হয়?

Address প্রিন্ট করতে হলে Variable এর সামনে address operator ইউজ করতে হবে।
address operator হলো → &

X এর Memory address location প্রিন্ট করতে হলে &X লিখতে হবে
এইভাবে,

printf(“ %d ”, &x);

```
main()
{
    int x  = 5;
    printf(“ %d \n”, x);  //value of x
    printf(“ %d ”, &x);  //memory address of x in Integer (%d) format
}
```

```c
main()
{
    int x  = 5;
    printf(“ %d \n”, x);  // x এর ভ্যালু 5 প্রিন্ট হবে
    printf(“ %d ”, &x);  //memory address of x in Integer (%d) format
}
```

memory address মূলত বাইনারিতে স্টোর হয়, কিন্তু বাইনারিতে দেখানোর জন্য কোনো format specifier such as (%b), C programming language এ নাই, তাই Integer (%d) format এ প্রিন্ট করে দেখানো হয়েছে, তবে অক্টাল , হেক্সাডেসিমেল এইসব format specifier আছে

যদি, `printf(“ %o ”, &x);  //memory address of x in Octal (%o) format`
`printf(“ %x ”, &x);  //memory address of x in Hexadecimal (%x) format`

```c
int main(){
    int cse = 18;
    ///value of cse
    printf("%d \n", cse);
    ///address of cse
    printf("%d \n", &cse);
    printf("%o \n", &cse);
    printf("%x \n", &cse);
}
```

```
output:
18
6422300
30377434
61ff1c
```

❑ এখন আসি Pointer কি জিনিস?

Definition:- The pointer is a variable which stores the address of another variable.

পয়েন্টার হচ্ছে এমন একটা ভ্যারিয়েবল যেটা অন্য ভ্যারিয়েবলে এর মেমোরি এড্রেস স্টোর করতে ব্যবহার করা হয়
নিচের কোডটি লক্ষ্য করি,

```
main(){
    int x = 8 ;
    int hello = &x;
}
```

এখানে x এর মেমোরি এড্রেস hello ভ্যারিয়েবল এ স্টোর করেছি [hello = &x]

তাহলে hello কি একটা পয়েন্টার ভ্যারিয়েবল?
উত্তরঃ- না, hello তে x এর মেমোরি এড্রেস স্টোর করলেও hello একটা সাধারণ ভ্যারিয়েবল।

তাহলে পয়েন্টার ভ্যারিয়েবল কিভাবে ডিক্লেয়ার করা হয়ে থাকে?
উত্তরঃ- একটা ভ্যারিয়েবলকে পয়েন্টার ভ্যারিয়েবল হতে হলে এর সামনে অবশ্যই **পয়েন্টার(*)** চিহ্ন দিতে হবে।

নিচের কোডটি লক্ষ্য করি,

```
main(){
    int x = 8 ;
    int *hello = &x;
}
```

কোডটি লক্ষ্য করি →

```
main(){
        int x = 8 ; //normal variable
        int hello1 = &x; //normal variable
        int *hello2 = &x //pointer variable
        printf("%d \n", x);
        printf("%d \n", hello1);
        printf("%d \n", hello2);
}
```

```
Output:
8
6422292
6422292
```

উভয় ক্ষেত্রেই মেমোরি এড্রেস লোকেশন সেইম, [6422292]
তাহলে Normal Variable আর Pointer Variable এর মধ্যে পার্থক্য কোথায়?

Pointer Variable দিয়ে
1. X এর Value এবং
2. X এর Memory Adddress দুইটাই বের করা যায়

কিভাবে বের করা যায়?

প্রিন্ট করার সময় পয়েন্টার চিহ্ন দিলেই ভ্যালু প্রিন্ট হয়। এইভাবেঃ-

```
int x = 8;
int *hello2 = &x;
```

শুধু hello2 মেমোরি এড্রেস প্রিন্ট করবে
`printf("Address of X : %d \n", hello2);`

কিন্তু *hello2 ভ্যালু প্রিন্ট করবে
`printf("Value of X : %d \n", *hello2);`

আর Normal Variable দিয়ে শুধু Memory Adddress বের করা যায়

নিচের কোডটি লক্ষ্য করি,

```
int main(){
    int x = 8;
    printf("x : %d \n", x);

    int hello1 = &x;
    printf("Address of X : %d \n", hello1);
    //printf("Address of X : %d \n", *hello1); → Wrong

    int *hello2 = &x;
    printf("Address of X : %d \n",  hello2);
    printf("Value of X : %d \n", *hello2);
}
```

```
x : 8
Address of X : 6422292
Address of X : 6422292
Value of X : 8
```

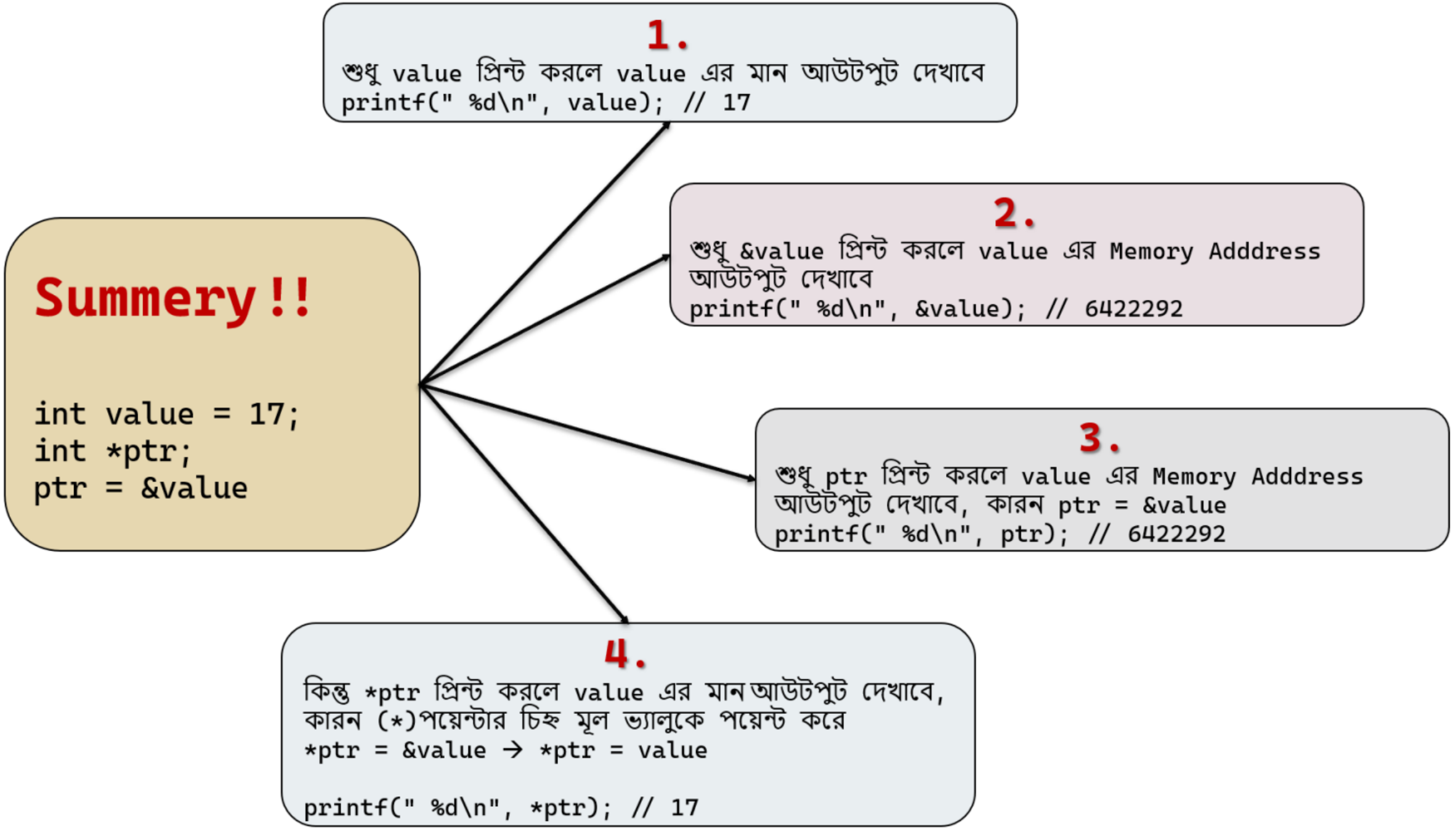
Explanation:-

1. প্রথমে X এর ভ্যালু প্রিন্ট হয়েছে → 8

2. hello1 নামের একটি সাধারন ভ্যারিয়েবল এ X এর মেমোরি এড্রেস লোকেশন রাখলাম এবং hello1 কে প্রিন্ট করার পর X এর মেমোরি এড্রেস 6422292 পেয়েছি

3. যেহেতু hello1 একটি সাধারন ভ্যারিয়েবল তাই একে যদি পয়েন্টার চিহ্ন দিয়ে প্রিন্ট করতে চাই CodeBlocks এ Error দেখাবে। অর্থাৎ * hello1 → ভুল

4. *hello2 নামের একটা Pointer Variable Declare করলাম, প্রথমে hello2 লিখে Memory Adddress → 6422292 প্রিন্ট করলাম, এরপরে *hello2 তে যার(X) ভ্যালু রাখা তার ভ্যালু → 8 প্রিন্ট করলাম


Summery !!
int value = 17;
int *ptr;
ptr = &value
1.
শুধু value প্রিন্ট করলে value এর মান আউটপুট দেখাবে
printf(" %d\n", value); // 17
2.
শুধু &value প্রিন্ট করলে value এর Memory Adddress আউটপুট দেখাবে
printf(" %d\n", &value); // 6422292
3.
শুধু ptr প্রিন্ট করলে value এর Memory Adddress আউটপুট দেখাবে, কারন ptr = &value
printf(" %d\n", ptr); // 6422292
4.
কিন্তু *ptr প্রিন্ট করলে value এর মান আউটপুট দেখাবে, কারন (*)পয়েন্টার চিহ্ন মূল ভ্যালুকে পয়েন্ট করে
*ptr = &value → *ptr = value
printf(" %d\n", *ptr); // 17

একটা পয়েন্টার দিয়ে একের অধিক ভেরিয়েবলকে পয়েন্ট করা যায়

Application – 1:- Pointer pointing to different variable

```
main(){
    int x = 5;
    int y = 6;
    int z = 8;
    int *mk;
    mk =  &x;
    printf("X  = %d\n", *mk);

    mk =  &y;
    printf("X  = %d\n", *mk);

    mk =  &z;
    printf("X  = %d\n", *mk);
}
```

যেমন আমরা বিভিন্ন ভেরিয়েবলকে [x,y,z] একটা ভেরিয়েবল[mk] এ copy করে রাখতে পারি।

```
main()
{
    int x = 5;
    int y = 6;
    int z = 8;
    int mk;
    mk =  x;
    printf("X  = %d\n", mk);

    mk =  y;
    printf("X  = %d\n", mk);

    mk =  z;
    printf("X  = %d\n", mk);
}
```

দুইটা পয়েন্টার ভ্যারিয়েবল দিয়ে যোগ-বিয়োগ [+ , -] করা যায়।

Application – 2:- Sum and Sub of two pointer

```
main(){
    int a = 2;
    int b = 3;

    int *p = &a;
    int *q = &b;

    int sum = *p + *q;
    int sub  = *p - *q;
    printf("sum  = %d\n", sum);
    printf("sub  = %d\n", sub);
}
```

Output: ??

পয়েন্টার N.B.:-

1. (*)পয়েন্টার চিহ্ন দিলেই ভ্যারিয়েবলের ভ্যালুকে রেফার করবে।
2. (*)পয়েন্টার চিহ্ন না দিলে ভ্যারিয়েবলের এড্রেসকে রেফার করবে।
3. দিয়ে গুন বা ভাগ করলে কি করলে কি কোনো সমস্যা হবে?

ঠিক যেভাবে আমরা নরমাল যোগ বিয়োগ করি।

```
main(){
    int a = 2;
    int b = 3;

    int p = a;
    int q = b;

    int sum = p + q;
    int sub  = p - q;
    printf("sum  = %d\n", sum);
    printf("sub  = %d\n", sub);
}
```

Output: ??

❑ REVISE Important Note:-

→ Attention:- প্রত্যেকটা ভ্যারিয়েবলে এর একটা করে এড্রেস থাকে মেমোরিতে। হোক সেটা পয়েন্টার ভ্যারিয়েবল বা সাধারন ভ্যারিয়েবল!!

উদাহরণ এর মাধ্যমে বুঝিঃ-

Example – 1: আগের একটা উদাহরণ আবার একটু দেখিঃ-

```
int main()
{
    int x = 5, y = 6, z = 7;
}
```

Memory Location of Variable	Value
1111101010110001	101
1100101010110001	110
0001101010110001	111

এখানে প্রতিটি ভ্যারিয়েবল এর জন্যে [x , y, z] একটি করে মেমোরি এড্রেস দেওয়া আছে,
এবং এই এড্রেস গুলোতে তাদের corresponding value থাকে,
অর্থাৎ
x এর Memory Address Location হলো 1111111011010010001, এই লোকেশন এ 5 [101] ভ্যালু স্টোর করা আছে।
y এর Memory Address Location হলো 1100101011010010001, এই লোকেশন এ 6 [110] ভ্যালু স্টোর করা আছে।
z এর Memory Address Location হলো 0001101011010010001, এই লোকেশন এ 7 [111] ভ্যালু স্টোর করা আছে।

আরও একট উদাহরণ দেখিঃ-

```
Let,
int it = 3;
int *pass = &it;
```

এই ক্ষেত্রে কি হবে?
যেহেতু it একটা ভ্যারিয়েবল, অবশ্যই it এর একটা Memory Address Location থাকবে যেখানে এর ভ্যালু 3[011] বাইনারি ফরমেট আকারে স্টোর থাকবে,
এখন কথা হচ্ছে pass ও তো একটা ভ্যারিয়েবল, তাই না? হোক সেটা পয়েন্টার ভ্যারিয়েবল। তাহলে এর জন্যেও তো Memory তে একটা Address Location Assign করা থাকতে হবে, Right?
আমরা এতোক্ষণ দেখেছি it, x , y, z এরা মোমোরিতে কোন লোকেশনে থাকে। **কিন্তু প্রশ্ন হলো এই পয়েন্টার ভ্যারিয়েবল মোমোরির কোথায় থাকে?**
আমরা তো জেনেই এসেছি একটা ভ্যারিয়েবল এর Memory Address Location লোকেশন কিভাবে বের করতে হবে, পয়েন্টার ভ্যারিয়েবল এর ক্ষেত্রেও ঠিক একই ভাবে বের করতে হবে।
অর্থাৎ, উপরের উদাহরনে,
it এর Memory Address Location বের করা যায় এইভাবে → `printf("%d", &it);`
অথবা, এইভাবে → `printf("%d", pass);` কারন `pass = &it`
একই ভাবে,
pass এর Memory Address Location বের করা যায় এইভাবে → `printf("%d", &pass);`
অথবা, আরেকটা পয়েন্টার ভ্যারিয়েবল এ রেখেঃ-

```
int it = 3;
int *pass = &it;
int *newp = &pass;
printf("%d", newp); //address of pointer variable ‘pass’
```

এই ব্যাপারে আমরা বিস্তারিত দেখবো পরের Example এ.

Example - 1: Visually

```
int it = 3;
int *pass = &it;
int *newp = &pass;
printf("%d", it);
printf("%d", pass);
printf("%d", newp);
```

	Memory Address Location	Value	
it →	11111000	00000011	= 3
pass →	11111001	11111000	= &it
Newp →	11111011	11111001	= &pass

Example – 2: এই কোডটি লক্ষ্য করি,

```
int main(){
    int x = 17;
    int y = &x;

    printf("Value of X  = %d\n", x);
    printf("Address of X = %p\n", &x);
    printf("Address of X in Y  = %d\n", y);

    int *p = &x;
    printf("Value of X in P = %d\n", *p);
    printf("Address of X in P = %d\n", p);
    printf("Address of P = %d\n", &p);
}
```

Output:

```
Value of X  = 17
Address of X = 0061FF18
Address of X in Y  = 6422296

Value of X in P = 17
Address of X in P = 6422296
Address of P = 6422292
```

Explanation:-

1. X এর ভ্যালু প্রিন্ট

2. X এর এড্রেস প্রিন্ট → &x

3. X এর এড্রেস y এর মধ্যে স্টোর করে প্রিন্ট

4. পয়েন্টার ভ্যারিয়েবল [*p] দিয়ে ভ্যালু প্রিন্ট

5. X এর এড্রেস p এর মধ্যে স্টোর করে প্রিন্ট

6. P ও একটা ভ্যারিয়েবল, এর এড্রস কিভাবে বের করব? → &p

Example – 3: Memory Address Location decimal[%d] ছাড়াও hexa-decimal[%x] বা octal[%o] ফরমেটে প্রিন্ট করে দেখানো যায়। তবে বাইনারিতে দেখানোর জন্যে কোনো format specifier নাই।

```
int main(){

    ///ex-3
    int x = 17;
    int *p = &x;

    printf("Address of X in octal = %o \n", &x);
    printf("Address of X at P in octal = %o \n\n", p);

    printf("Address of X in hexadecimal = %x \n", &x);
    printf("Address of X at P in hexadecimal = %x \n", p);
}
```

Output:

```
Address of X in octal = 30377430
Address of X at P in octal = 30377430

Address of X in hexadecimal = 61ff18
Address of X at P in hexadecimal = 61ff18
```

Example – 4: Pointers Of Pointers

```
int main(){

    int x = 51;
    printf("Value of X  = %d\n",  x);
    printf("Addres of X  = %d\n\n", &x);

    int *y = &x;

    printf("Addres of X in Y = %d\n",  y);
    printf("Value of Y = %d\n", *y);
    printf("Address of Y  = %d\n\n", &y);

    int **z = &y;
    printf("Address of Y in Z  = %d\n", z);
    printf("Value of Z  = %d\n", **z);
    printf("Address of Z  = %d\n\n", &z);

    int ***w = &z;
    printf("Address of Z in W  = %d\n", w);
    printf("Value of W  = %d\n",  ***w);
    printf("Address of W  = %d\n", &w);
}
```

Explanation:-

1. এইটুকু অংশ কিভাবে কাজ করে সেটা আমরা অলরেডি Example – 2 তে দেখে এসেছি।

2. কিন্তু এখন **z আবার কি জিনিস? এইটা কিভাবে কাজ করে?

int **z আসলে x কে রেফার করে, তাই **z প্রিন্ট করলে x = 51 প্রিন্ট হয়।
এটা কিভাবে কাজ করে?

মনে রাখার জন্য একটা বীজগণিতের শর্টকাট ট্রিক্স দেখিঃ- [কু-বুদ্ধি]

int **z = &y [দেওয়া আছে]
int **z = &(&x)
int **z = x [& এবং & কাটাকাটি]

কারন int *y = &x,
এক অন্যভাবেও লিখা যায়
int *y;
y = &x;
তাই y এর মান &x
বসানো হয়েছে

3. তাহলে ***w প্রিন্ট করলে কি আউটপুট আসবে?
int ***w = &z [দেওয়া আছে]
int ***w = &(&y)
int ***w = &(&(&x)) [int *y = &x]
int ***w = x [& এবং & এবং & কাটাকাটি]

কারন int **Z = &y
এক অন্যভাবেও লিখা যায়
int **z;
z = &y;
তাই z এর মান &z
বসানো হয়েছে

❑C – Program to illustrate Call-By-Value and Call-By-Reference

Call-By-Value

```
#include <stdio.h>
void swap(int a, int b)
{
    int temp = a;
    a = b;
    b = temp;
}
int main()
{
    int x = 10, y = 20;
    printf("before swap are: %d, %d\n", x, y);
    swap(x, y);
    printf("after swap are: %d, %d", x, y);
}
```

Output:

```
before swap are: 10, 20
after swap are: 10, 20
```

Call-By-Reference

```
#include <stdio.h>
void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main()
{
    int num1 = 10, num2 = 20;
    printf("Before swapping: num1 = %d, num2 = %d\n", num1, num2);
    swap(&num1, &num2);
    printf("After swapping: num1 = %d, num2 = %d\n", num1, num2);
}
```

Output:

```
Before swapping: num1 = 10, num2 = 20
After swapping: num1 = 20, num2 = 10
```

❑C – Program to Swap Two variable using pointer and function

```
#include <stdio.h>
void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main()
{
    int num1 = 10, num2 = 20;
    printf("Before swapping: num1 = %d, num2 = %d\n", num1, num2);
    swap(&num1, &num2);
    printf("After swapping: num1 = %d, num2 = %d\n", num1, num2);
}

Output:
Before swapping: num1 = 10, num2 = 20
After swapping: num1 = 20, num2 = 10
```

❑ C – Program to access array elements using pointers

```
#include <stdio.h>

int main() {
    int a[5] = {10, 20, 30, 40, 50};
    int *ptr, i;
    ptr = &a[0];

    for(i=0; i<5; i++){
        printf("%d ", *ptr);
        ptr++;
    }
}
```

```
Output:
10 20 30 40 50
```

❑ What is a Pointer in C?

A pointer is defined as a derived data type that can store the address of other C variables or a memory location. We can access and manipulate the data stored in that memory location using pointers.

Pointers are one of the core components of the C programming language. A pointer can be used to store the memory address of other variables, functions, or even other pointers. The use of pointers allows low-level memory access, dynamic memory allocation, and many other functionality in C.

❑ Syntax of C Pointers

The syntax of pointers is similar to the variable declaration in C, but we use the (*) dereferencing operator in the pointer declaration.

```
datatype * ptr;
```

Where,

- ptr is the name of the pointer.
- datatype is the type of data it is pointing to.

The above syntax is used to define a pointer to a variable. We can also define pointers to functions, structures, etc.

❑ How to Use Pointers?

The use of pointers in C can be divided into three steps:

1. Pointer Declaration
2. Pointer Initialization
3. Pointer Dereferencing

1. Pointer Declaration

In pointer declaration, we only declare the pointer but do not initialize it. To declare a pointer, we use the (*) dereference operator before its name.

Example:- int *ptr;

The pointer declared here will point to some random memory address as it is not initialized. Such pointers are called wild pointers.

2. Pointer Initialization

Pointer initialization is the process where we assign some initial value to the pointer variable. We generally use the (&: ampersand) addressof operator to get the memory address of a variable and then store it in the pointer variable.

Example:-

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

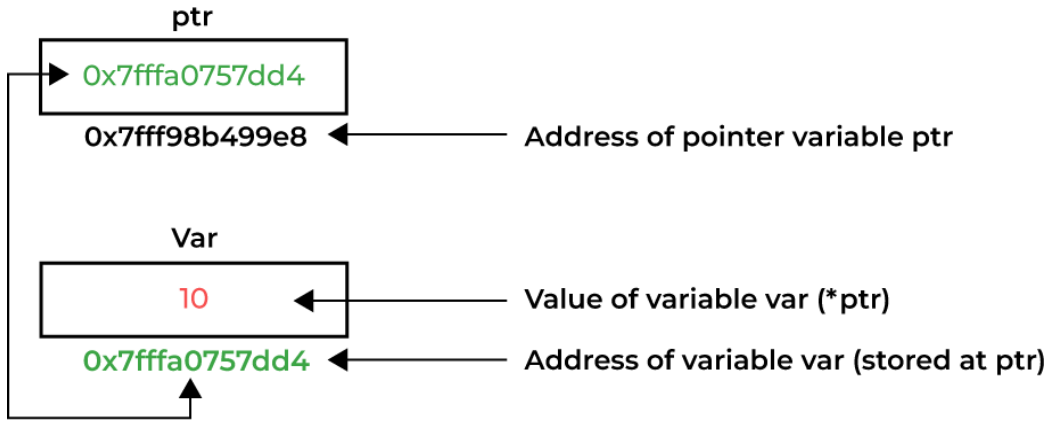
We can also declare and initialize the pointer in a single step. This method is called pointer definition as the pointer is declared and initialized at the same time.

Example:- int *ptr = &var;

Note: It is recommended that the pointers should always be initialized to some value before starting using it. Otherwise, it may lead to number of errors.

3. Pointer Dereferencing

Dereferencing a pointer is the process of accessing the value stored in the memory address specified in the pointer.
We use the same (*) dereferencing operator that we used in the pointer declaration.




```c
// C program to illustrate Pointers
#include <stdio.h>
void geeks(){
     int var = 10;
     // declare pointer variable
     int* ptr;
     // note that data type of ptr and var must be same
     ptr = &var;
     // assign the address of a variable to a pointer
     printf("Value at ptr = %p \n", ptr);
     printf("Value at var = %d \n", var);
     printf("Value at *ptr = %d \n", *ptr);
}
int main(){
    geeks();
}
```

Output:

```
Value at ptr = 0x7ffca84068dc
Value at var = 10
Value at *ptr = 10
```

❑ Various Types of Pointers in C

1. Integer Pointers

As the name suggests, these are the pointers that point to the integer values.

Syntax:- int *ptr;

These pointers are pronounced as Pointer to Integer.

2. Array Pointer

Pointers and Array are closely related to each other. Even the array name is the pointer to its first element. They are also known as Pointer to Arrays. We can create a pointer to an array using the given syntax.

Syntax:- char *ptr = &array_name;

3. Double Pointers

In C language, we can define a pointer that stores the memory address of another pointer. Such pointers are called double-pointers or pointers-to-pointer. Instead of pointing to a data value, they point to another pointer.

Syntax:-

datatype */ pointer_name;

➢ Other pointers are Structure Pointer, Function Pointers, NULL Pointer, Void Pointer, Wild Pointers, Constant Pointers, Pointer to Constant, Dangling pointer etc.

❑ C Pointer Arithmetic

The Pointer Arithmetic refers to the legal or valid arithmetic operations that can be performed
on a pointer. These operations include:

- Increment in a Pointer
- Decrement in a Pointer
- Addition of integer to a pointer
- Subtraction of integer to a pointer
- Subtracting two pointers of the same type
- Comparison of pointers of the same type.
- Assignment of pointers of the same type.

❑ Uses of Pointers in C

The C pointer is a very powerful tool that is widely used in C programming to perform various useful operations. It is used to achieve the following functionalities in C:

1. Pass Arguments by Reference
2. Accessing Array Elements
3. Return Multiple Values from Function
4. Dynamic Memory Allocation
5. Implementing Data Structures
6. In System-Level Programming where memory addresses are useful.
7. In locating the exact value at some memory location.
8. To avoid compiler confusion for the same variable name.
9. To use in Control Tables.

❑ What are the differences between an array and a pointer?

Pointer	Array
A pointer is a derived data type that can store the address of other variables.	An array is a homogeneous collection of items of any type such as int, char, etc.
Pointers are allocated at run time.	Arrays are allocated at runtime.
The pointer is a single variable.	An array is a collection of variables of the same type.
Dynamic in Nature	Static in Nature.

❑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.

❑ Relationship between Arrays and Pointers

Arrays and Pointers are closely related to each other such that we can use pointers to perform all the possible operations of the array. The array name is a constant pointer to the first element of the array and the array decays to the pointers when passed to the function.

```
#include <stdio.h>
int main(){
    int arr[5] = { 10, 20, 30, 40, 50 };
    int* ptr = &arr[0];

    printf("Address Stored in Array name: %p\nAddress of 1st Array Element: %p\n", arr, &arr[0]);

    printf("Array elements using pointer: ");
    for (int i = 0; i < 5; i++) {
        printf("%d ", *ptr++);
    }
}
```

Output
```
Address Stored in Array name: 0x7ffcab67d8e0
Address of 1st Array Element: 0x7ffcab67d8e0
Array elements using pointer: 10 20 30 40 50
```

❑ Exercise problems:

```c
#include <stdio.h>
int foo(int* a, int* b){
    int sum = *a + *b;
    *b = *a;
    return *a = sum - *b;
}
int main(){
    int i = 0, j = 1, k = 2, l;
    l = i++ || foo(&j, &k);
    printf("%d %d %d %d", i, j, k, l);
}
```

```c
int main(){
      int a=1,b=2,c=3;
      int *p,*q,**r;
      p=&a;
      r=&q;
      q=&c;
      a=*q+**r;
      printf("x=%d y=%d z=%d\n",**r,*p,*q);
      *r=p;
      a=*q+**r;
      printf("w=%d\n",a);
}
```

```c
#include <stdio.h>
void f1 (int a, int b){
  int c;
  c=a; a=b; b=c;
}
void f2 (int *a, int *b){
  int c;
  c=*a; *a=*b;*b=c;
}
int main(){
  int a=4, b=5, c=6;
  f1(a, b);
  f2(&b, &c);
  printf (“%d”, c-a-b);
}
```

```c
#include <stdio.h>
int main( ){
      int a[3][4],i,j;
      for(i=0;i< 3;i++)
      for(j=0;j<4;j++)
      a[i][j]=i+2*j;
      printf(“%d %d %d \n”,*a[2]+2,*(a[2]+2),*(a[1]+6));
      printf(“%d %d %d \n”,*(*a+3),**(a+3),*(&a[1][3]+5));
}
```

```c
printf("%d,%d,%d\n",sizeof(char*),sizeof(int*),sizeof(float*)) ;
```

```c
#include <stdio.h>
int main(){
    int arr[5];
    printf("%u %u", arr + 1, &arr + 1);
}
```

❑ Exercise problems:

```
int a = 14, b = 8, c = 21;
int max = (a > b) ? ((a > c) ? a : c) : ((b > c) ? b : c);
const char* result = (max % 2 == 0) ? "Even" : "Odd";
printf("Maximum value: %d, which is %s", max, result);
```

```
int a = -3, b = 12, c = -8;
int middle = (a > b) ? ((b > c) ? b : (a > c ? c : a)) : ((a > c) ? a : (b > c ? c : b));
const char* result = (middle > 0) ? "Positive" : "Negative";
printf("Middle value: %d, which is %s", middle, result);
```

```
int x = 13, y = 30, z = 20;
int max = (x > y) ? ((x > z) ? x : z) : ((y > z) ? y : z);
const char* result = (max % 5 == 0) ? "Divisible by 5" : "Not divisible by 5";
printf("Maximum value: %d, %s", max, result);
```

```
#include <stdio.h>
int main() {
      int int array[3] = {4,9,3};
      int *ptr1,*ptr2,*ptrs[3];
      ptr1 = &int array[2];
      ptr2 = &int array[0];
      ptrs[0]=ptr1;
      ptrs[1]=ptr2;
      ptrs[2]=&int array[1];
      printf(“The values are %d %d %d\n”,*ptrs[0],*ptrs[1],*ptrs[2]);
}
```

```
#include<stdio.h>
int main(){
      char *ptr;
      char aChars[10] = "Excellent";
      ptr = aChars;
      printf("Word = %s\n", aChars);
      *ptr = ’V’;
      printf("Now word = %s\n", aChars);
      printf("Mystery char = %c\n",*ptr+3);
      printf("Mystery char = %c\n",*(ptr+3));
      *(ptr+1) =’ ’;
      *(ptr+3) =’ ’;
      printf("String = %s\n",ptr);
}
```