CSE101-Lec# 18,19

- Pointers in C

Introduction-Pointer declaration and Initialization

- A pointer is a variable that holds the address of another variable.
- The general syntax of declaring pointer variable is

  data_type *ptr_name;

  Here, data_type is the data type of the value that the pointer will point to. For example:

  int *pnum;char *pch; float *pfnum; //Pointer declaration

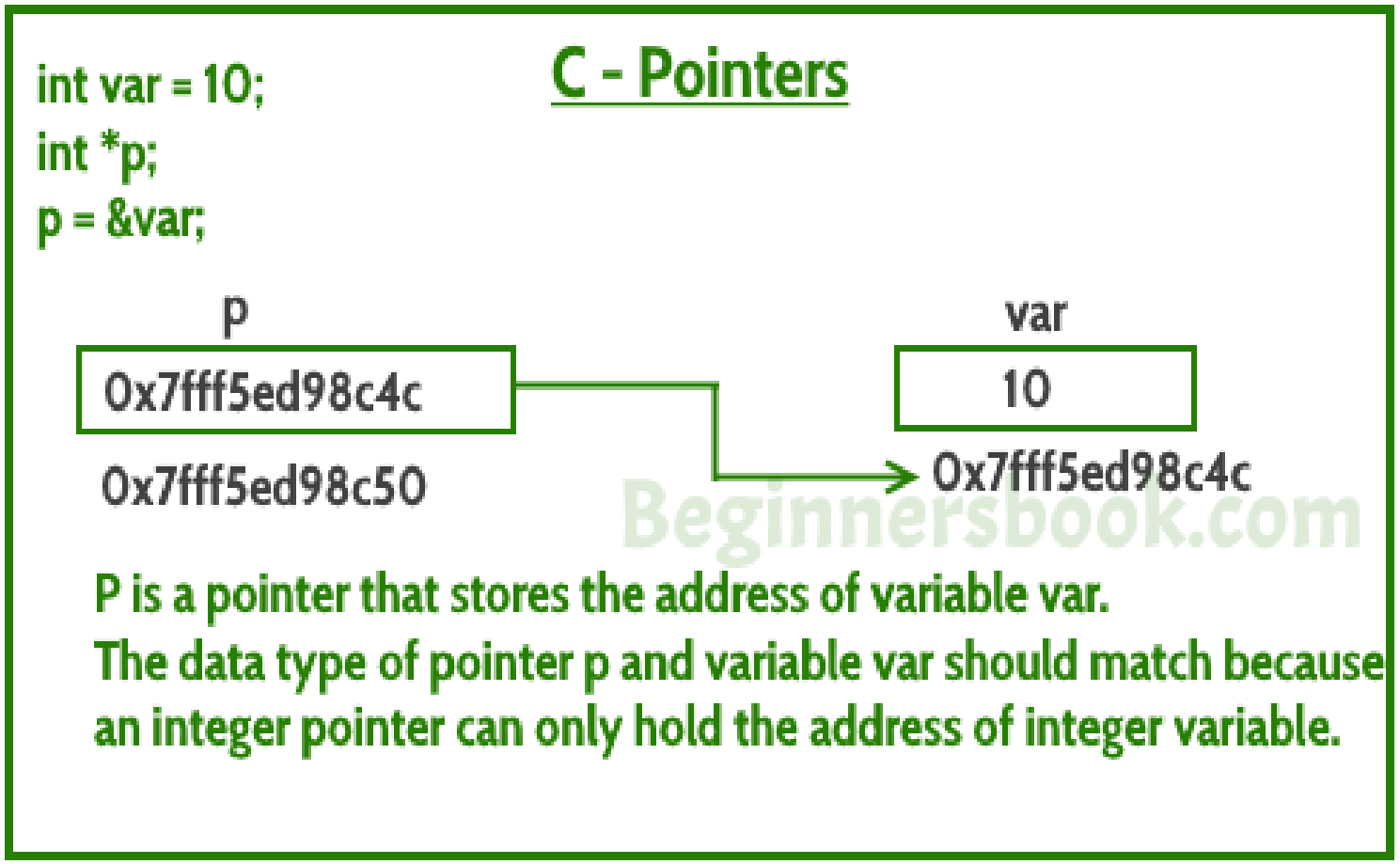
  int x= 10;

  int *ptr = &x; //Pointer initialization[When some variable's address is assigned to pointer, it is said to be initialized]

  The '*' informs the compiler that ptr is a pointer variable and the int specifies that it will store the address of an integer variable. ['*' is also known as indirection/ or dereferencing/ or value at address operator]

  The & operator retrieves the address of x, and copies that to the contents of the pointer ptr. ['&' is also known as address of operator]

Understanding pointers

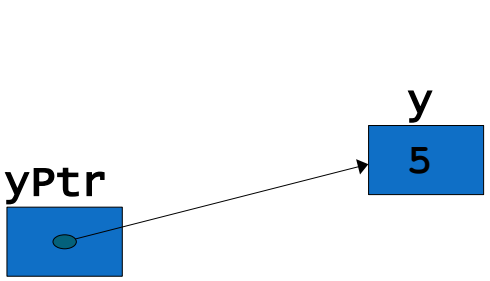


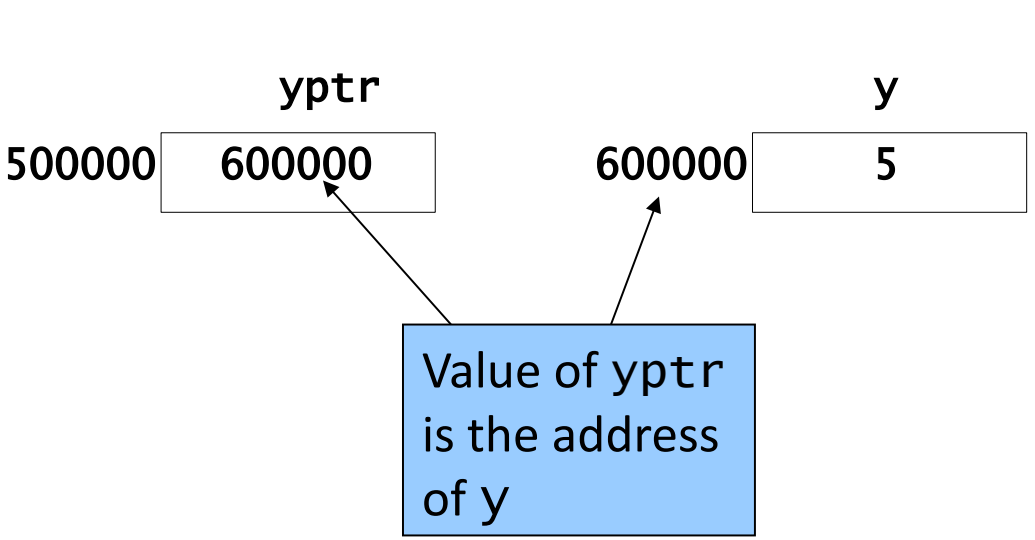
Pointer Operators

- & (address operator)
  - Returns address of operand

```
int y = 5;
int *yPtr;
yPtr = &y;      /* yPtr gets address of y */
```

yPtr "points to" y





Pointer Operators

- * (indirection/dereferencing operator)
  - Returns the value of the variable that it points to.
  - `*yptr` returns value of `y` (because `yptr` points to `y`)
  - * can be used for assignment

```
*yptr = 7;  /* changes y to 7 */
```

Example Code

```c
#include <stdio.h>

int main()
{
   int a;        /* a is an integer */
   int *aPtr;    /* aPtr is a pointer to an integer */

   a = 7;
   aPtr = &a;    /* aPtr set to address of a */

   printf( "The address of a is %p"
           "\nThe value of aPtr is %p", &a, aPtr );

   printf( "\n\nThe value of a is %d"
           "\nThe value of *aPtr is %d", a, *aPtr );

   printf( "\n\nShowing that * and & are complements of "
           "each other\n&*aPtr = %p"
           "\n*&aPtr = %p\n", &*aPtr, *&aPtr );

   return 0; /* indicates successful termination */

} /* end main */
```

This program demonstrates the use of the pointer operators: & and *

Output

```
The address of a is 0012FF7C
The value of aPtr is 0012FF7C

The value of a is 7
The value of *aPtr is 7

Showing that * and & are complements of each other.
&*aPtr = 0012FF7C
*&aPtr = 0012FF7C
```

Key points related to pointers

- ***Data type of the pointer variable and variable whose address it will store must be of same type***

Example:

int x=10;

float y=2.0;

int *px=&y;//Invalid, as px is of integer type and y is of float type

int *ptr=&x;//Valid as both ptr and x are of same types

- ***Any number of pointers can point to the same address***

Example:

int x=12;

int *p1=&x,*p2=&x,*p3=&x;// All the three pointers are pointing towards x

- ***Memory taken by any kind of pointer(i.e int, float, char, double…) as always equivalent to the memory taken by unsigned integer, as pointer will always store address of a variable(which is always unsigned integer), so the type of pointer will not make any difference***

Example-size taken by different type of pointers

```
#include<stdio.h>
int main()
{
        int *pnum;
        char *pch;
        float *pfnum;
        double *pdnum;
        long *plnum;
        printf("\n Size of integer pointer=%d",sizeof(pnum));
        printf("\n Size of character pointer=%d",sizeof(pch));
        printf("\n Size of float pointer=%d",sizeof(pfnum));
        printf("\n Size of double pointer=%d",sizeof(pdnum));
        printf("\n Size of long pointer=%d",sizeof(plnum));
        return 0;
}
//All will give the same answer(equivalent to size taken by unsigned integer for a particular compiler)
```

Program example-Finding area of circle using pointers

```
#include<stdio.h>
int main()
{
        double radius,area=0.0;
        double *pradius=&radius,*parea=&area;
        printf("\n Enter the radius of the circle:");
        scanf("%lf",pradius);
        *parea=3.14*(*pradius)*(*pradius);
        printf("\n The area of the circle with radius %.2lf = %.2lf",*pradius,*parea);
        return 0;
}
```

Program example-Factorial of a number using pointer

```
#include<stdio.h>
int main()
{
        int i,n,fact=1;
        int *pn,*pfact;
        pn=&n;
        pfact=&fact;
        printf("\n Enter number:");
        scanf("%d",pn);
        for(i=1;i<=*pn;i++)
        {
                *pfact=*pfact*i;
        }
        printf("\n Factorial of number is:%d",*pfact);
        return 0;
}
```

Program example-Reverse of a number using pointers

```
#include <stdio.h>
int main()
{
   int n, reversedNumber = 0, remainder;
   int *pn,*prn,*pr;
   pn=&n;
   prn=&reversedNumber;
   pr=&remainder;
   printf("Enter an integer: ");
   scanf("%d", pn);
   while(*pn != 0)
   {
      *pr = *pn%10;
      *prn = *prn*10 + *pr;
      *pn = *pn/10;
   }
   printf("Reversed Number = %d",*prn);

   return 0;
}
```

Types of pointers

- Null pointer
- Wild pointer
- Generic pointer(or void) pointer
- Constant pointer
- Dangling pointer

Null pointer

- A Null Pointer is a pointer that does not point to any memory location
- It is used to initialize a pointer variable when the pointer does not point to a valid memory address.
- So, if we don't know in the initial phases, where the pointer will point? , it is better to initialize pointer with NULL address

To declare a null pointer you may use the predefined constant NULL,

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

or

```
int *ptr=0;
```

We can overwrite the NULL address hold by NULL pointer with some valid address also, in the later stages of program

Note: It is invalid to dereference a null pointer.

Example

```
#include<stdio.h>
int main()
{
        int *ptr=NULL;
        int a=10;
        printf("%u",ptr);// 0 will be displayed
        printf(“%d”,*ptr);//Invalid(Dereferencing), as ptr is NULL at this point.
        ptr=&a;
    printf("\n%d",*ptr);//Now it is allowed, as NULL pointer has starting pointing somewhere
        return 0;
}
```

Wild pointer

- Pointer which are not initialized during its definition holding some junk value(or Garbage address) are Wild pointer.
- Example of wild pointer:

```
int *ptr;
```

- Every pointer when it is not initialized is defined as a wild pointer.
- As pointer get initialized, start pointing to some variable its defined as pointer, not a wild one.

Example

```
#include<stdio.h>
int main()
{
        int *ptr;//Wild pointer
        int a=10;
        //printf("%u",ptr);//Gives garbage address value
        //printf("\n%d",*ptr);//Gives garbage value stored in the garbage address
        ptr=&a;//Now ptr is not a wild pointer
        printf("\n%d",*ptr);//
        return 0;
}
```

Void pointer

- Is a pointer that can hold the address of variables of different data types at different times also called generic pointer.
- The syntax for declaring a void pointer is

```
void *pointer_name;
```

- Here, the keyword **`void`** represents that the pointer can point to value of any data type.
- But before accessing the value through generic pointer by dereferencing it, it must be properly **typecasted**.
- To Print value stored in pointer variable:

```
*(data_type*) pointer_name;
```

Limitations of void pointers:

- void pointers cannot be directly dereferenced. They need to be appropriately typecasted.
- Pointer arithmetic cannot be performed on void pointers.

Example

```
#include<stdio.h>
int main()
{
        int x=10;
        char ch='A';
        void *gp;
        gp=&x;
        printf("\n Generic pointer points to the integer value=%d",*(int*)gp);
        gp=&ch;
        printf("\n Generic pointer now points to the character %c",*(char*)gp);
        return 0;
}
```

Constant Pointers

- A constant pointer, **`ptr`**, is a pointer that is initialized with an address, and cannot point to anything else.
- But we can use **`ptr`** to change the contents of variable pointing to
- Example

```
int value = 22;
int * const ptr = &value;
```

Constant Pointer

- Example:

  ```
  int * const ptr2
  ```

  indicates that ptr2 is a pointer which is constant. This means that ptr2 cannot be made to point to another integer.

- However the integer pointed by ptr2 can be changed.

Example

```
#include<stdio.h>
int main()
{
    int var1 = 60, var2 = 70;
    int *const ptr = &var1;
    printf("\n%d",*ptr);
    //ptr = &var2; //Invalid-Error will arise
    //printf("%d\n", *ptr);
    return 0;
}
```

Dangling pointer

- It is a type of pointer which point towards such a memory location which is already deleted/ or deallocated.
- It is a problem associated with pointers, where in a pointer is unnecessarily pointing towards deleted memory location
- It can be resolved through assigning NULL address once, the memory has been deallocated

Dangling pointer-Example 1[Compile time case]
When local variable goes out of scope

```
#include<stdio.h>
int main()
{
   int *ptr;
   {
      int val=23;
      ptr=&val;                    //val has scope within the block [Auto Storage Class]
      printf("\n%d",*ptr);// 23 is printed
      printf("\n%u",ptr);// Address of val is printed
   }
   printf("\n%u",ptr);// Same address is printed, even val is destroyed, hence ptr is dangling pointer
   ptr=NULL;//Solution
   printf("\n%u",ptr);// Now ptr is not a dangling pointer[0 address value is printed]
   return 0;
}
```

Dangling pointer-Example 2[Runtime/or Dynamic memory allocation case]
When free() function is called

```
// Deallocating a memory pointed by ptr causes
// dangling pointer
#include <stdlib.h>
#include <stdio.h>
int main()
{

        int n=1;
        int *ptr = (int *)malloc(n*sizeof(int));
        *ptr=6;
   printf("%d",*ptr);//6 is printed
   printf("\n%d",ptr);//Printing address hold by pointer before deallocation
   free(ptr);
   printf("\n%d",ptr);//Same address will be printed(Dangling pointer)
  //SOLUTION
  ptr = NULL;//Pointer is now changed to NULL pointer
  printf("\n%d",ptr);//0 will be printed
  return 0;
}
```

Example-1-Passing pointer to a function(or call by reference)

```
//Passing arguments to function using pointers
#include<stdio.h>
void sum(int *a,int *b,int *t);
int main()
{
        int num1,num2,total;
        printf("\n Enter the first number:");
        scanf("%d",&num1);
        printf("\n Enter the second number:");
        scanf("%d",&num2);
        sum(&num1,&num2,&total);
        printf("\n Total=%d",total);
        return 0;
}
void sum(int *a,int *b,int *t)
{
        *t=*a+*b;
}
```

Example-2-Passing pointer to a function(or call by reference)

```
#include<stdio.h>
void read(float *b,float *h);
void calculate_area(float *b,float *h,float *a);
int main()
{
        float base,height,area;
        read(&base,&height);
        calculate_area(&base,&height,&area);
        printf("\n Area is :%f",area);
        return 0;
}
void read(float *b,float *h)
{
        printf("\n Enter the base of the triangle:");
        scanf("%f",b);
        printf("\n Enter the height of the triangle:");
        scanf("%f",h);
}
void calculate_area(float *b,float *h,float *a)
{
        *a=0.5*(*b)*(*h);
}
```

Q1

What will be the output of the following C code?

```
#include <stdio.h>
int main()
{
    int *ptr, a = 10;
    ptr = &a;
    *ptr += 1;
    printf("%d,%d", *ptr, a);
}
```

A. 10,10

B. 10,11

C. 11,10

D. 11,11

Q2

Comment on the following pointer declaration.

int *ptr, p;

A. ptr is a pointer to integer, p is not

B. ptr and p, both are pointers to integer

C. ptr is a pointer to integer, p may or may not be

D. ptr and p both are not pointers to integer

Q3

What will be the output of the following C code?

```
#include <stdio.h>
int x = 0;
int main()
{
    int *ptr = &x;
    printf("%p\n", ptr);
    x++;
    printf("%p\n ", ptr);
}
```

A. Same address

B. Different address

C. Compile time error

D. None of these

Q4

```
#include <stdio.h>
   int main()
   {
      int x=10;
      int *p1=&x,*p2;
      *p1=x+3;
      p2=p1;
      *p2=*p1+2;
      printf("%d",x);
      return 0;
   }
```

A. 13

B. 12

C. 10

D. 15

Q5

What will be the output of the following C code?

```
#include <stdio.h>
int main()
{
    char *p = NULL;
    char *q = 0;
    if (p)
        printf(" p ");
    else
        printf("nullp");
    if (q)
        printf("q\n");
    else
        printf(" nullq\n");
}
```

a) nullp nullq

b) Nothing will be printed

c) Compile time error

Q6

What will be the output of the following C code?

```
#include <stdio.h>
int main()
{
    int i = 10;
    void *p = &i;
    printf("%d\n", (int)*p);
    return 0;
}
```

A. Compile time error

B. Program will crash

C. 10

D. Address of i

Q7

What will be the output of the following C code?

```
#include <stdio.h>
int main()
{
    int i = 10;
    void *p = &i;
    printf("%f\n", *(float*)p);
    return 0;
}
```

A. Compile time error

B. 10.000000

C. 10

D. 0.000000

Q8

What will be the output of the following C code?

```
#include <stdio.h>
int x = 0;
void main()
{
    int *const ptr = &x;
    printf("%p\n", ptr);
    ptr++;
    printf("%p\n ", ptr);
}
```

A. 0 1

B. Compile time error

C. 0xbfd605e8 0xbfd605ec

D. 0xbfd605e8 0xbfd605e8

Q9

What will be the output of the following C code?

```
#include <stdio.h>
void foo(int *p)
{
    int j = 2;
    p = &j;
    printf("%d ", *p);
}
int main()
{
    int i = 97, *p = &i;
    foo(&i);
    printf("%d ", *p);
}
```

A. 2 97

B. 2 2

C. Compile time error

D. Program will crash

Q10

What will be the output of the following C code?

```
#include <stdio.h>
void m(int *p, int *q)
{
    p=q;
    *p=8;
    *q=7;
}
int main()
{
    int a = 6, b = 5;
    m(&a, &b);
    printf("%d %d\n", a, b);
}
```

a) 8 7

b) 6 7

c) 6 5

d) 8 8