Pointers

Prepared by: Lec Tasmiah Tamzid Anannya, CS Dept, AIUB

Example

```
#include <stdio.h>
int main()
{
   int* pc, c;

   c = 22;
   printf("Address of c: %d\n", &c);
   printf("Value of c: %d\n\n", c);

   pc = &c;
   printf(“Address contained in pointer pc: %u\n", pc);
   printf(“Value of address contained in pointer pc: %d\n\n", *pc);

   c = 11;
   printf(“Address contained in pointer pc: %u\n", pc);
   printf(“Value of address contained in pointer pc: %d\n\n", *pc);

   *pc = 2;
   printf("Address of c: %u\n", &c);
   printf("Value of c: %d\n\n", c);
   return 0;
}
```

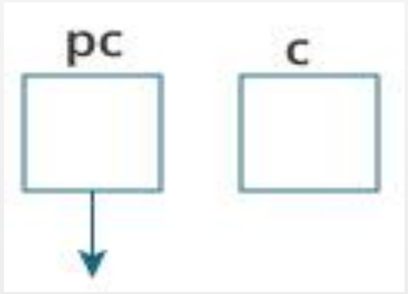
Example

```
#include <stdio.h>
int main()
{
   int* pc, c;

   c = 22;
   printf("Address of c: %d\n", &c);
   printf("Value of c: %d\n\n", c);

   pc = &c;
   printf(“Address contained in pointer pc: %u\n", pc);
   printf(“Value of address contained in pointer pc: %d\n\n", *pc);

   c = 11;
   printf(“Address contained in pointer pc: %u\n", pc);
   printf(“Value of address contained in pointer pc: %d\n\n", *pc);

   *pc = 2;
   printf("Address of c: %u\n", &c);
   printf("Value of c: %d\n\n", c);
   return 0;
}
```

```
Address of c: 6356744
Value of c: 22

Address contained in pointer pc: 6356744
Value of address contained in pointer pc: 22

Address contained in pointer pc: 6356744
Value of address contained in pointer pc: 11

Address of c: 6356744
Value of c: 2
```

Let’s see what actually happens

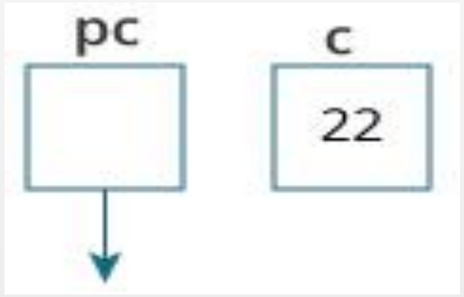
```
int *pc ,c;
```



Here, a pointer pc and a normal variable c, both of type int, is created. Since pc and c are not initialized at first, pointer pc points to either no address or a random address. And, variable c has an address but contains a random garbage value.

Let’s see what actually happens

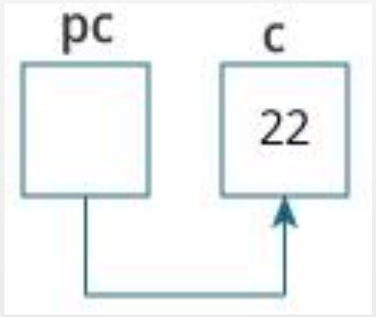
c=22;



22 is stored at the address of variable c.

Let’s see what actually happens

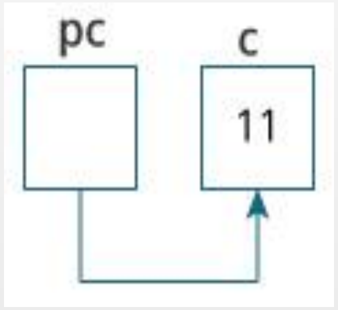
```
pc=&c;
```



This assigns the address of variable c to the pointer pc.
Here, the value of pc is same as the address of c.

Let's see what actually happens

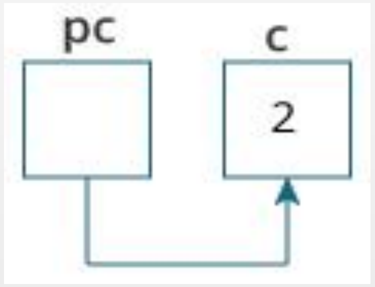
c=11;



This assigns 11 to variable c.
Since, pointer pc points to the same address as c, value pointed by pointer pc is 11 as well.

Let’s see what actually happens

*pc=2



This change the value at the memory location pointed by pointer pc to 2. Since the address of the pointer pc is same as the address of c,

value of c is also changed to 2.

Summary

int c	c	Value of the variable
	&c	Address of the variable
int *p	*p	Value of the address contained in the pointer
	p	Address that is contained in the pointer
	&p	Address of the memory location of the pointer

Common mistakes while working with pointer

- int c, *pc;

- // Wrong! pc is address whereas,
- // c is not an address.
- pc = c;

- // Wrong! *pc is the value pointed by address whereas,
- // &c is an address.
- *pc = &c;

- // Correct! pc is an address and,
- // &c is also an address.
- pc = &c;

- // Correct! *pc is the value pointed by address and,
- // c is also a value (not address).
- *pc = c;

Arrays and Pointer

```
#include <stdio.h>
int main()
{
   int x[4]={1,2,3,4};
   int i;

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

   printf("Address of array x: %u", x);
   return 0;
}
```

```
&x[0] = 6356732
&x[1] = 6356736
&x[2] = 6356740
&x[3] = 6356744
Address of array x: 6356732
```

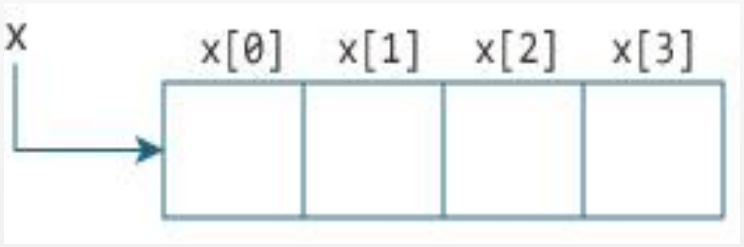
Arrays and Pointer

```
&x[0] = 6356732
&x[1] = 6356736
&x[2] = 6356740
&x[3] = 6356744
Address of array x: 6356732
```

- There is a difference of 4 bytes between two consecutive elements of array x. It is because the size of int is 4 bytes (on our compiler).
- Notice, that &x[0] and x gave us the same result.

Relation between Arrays and Pointers

- Suppose, int x[4] is an array.



- It is clear that, x and &x[0] contains the same address.
- So, x[0] and *x is equivalent.

Similarly,

- &x[1] is equivalent to x+1 and x[1] is equivalent to *(x+1).
- &x[2] is equivalent to x+2 and x[2] is equivalent to *(x+2).
- ...
- Basically, &x[i] is equivalent to x+i and x[i] is equivalent to *(x+i).

Example

```
#include <stdio.h>
int main()
{
  int i, x[6]={0,1,2,3,4,5};
  for(i=0;i<6;i++)
  {
      printf("%d ", x[i]);
  }
  return 0;
}
```

Example

```
#include <stdio.h>
int main()
{
  int i, x[6]={0,1,2,3,4,5};
  for(i=0;i<6;i++)
  {
      printf("%d ", x[i]);
  }
  return 0;
}
```

```
#include <stdio.h>
```

Example

```
#include <stdio.h>
int main()
{
  int i, x[6]={0,1,2,3,4,5};
  for(i=0;i<6;i++)
  {
      printf("%d ", x[i]);
  }
  return 0;
}
```

```
#include <stdio.h>
int main()
```

Example

```
#include <stdio.h>
int main()
{
  int i, x[6]={0,1,2,3,4,5};
  for(i=0;i<6;i++)
  {
      printf("%d ", x[i]);
  }
  return 0;
}
```

```
#include <stdio.h>
int main()
{
```

Example

```
#include <stdio.h>
int main()
{
  int i, x[6]={0,1,2,3,4,5};
  for(i=0;i<6;i++)
  {
      printf("%d ", x[i]);
  }
  return 0;
}
```

```
#include <stdio.h>
int main()
{
  int i, x[6]={0,1,2,3,4,5};
```

Example

```
#include <stdio.h>
int main()
{
  int i, x[6]={0,1,2,3,4,5};
  for(i=0;i<6;i++)
  {
      printf("%d ", x[i]);
  }
  return 0;
}
```

```
#include <stdio.h>
int main()
{
  int i, x[6]={0,1,2,3,4,5};
  for(i=0;i<6;i++)
```

Example

```
#include <stdio.h>
int main()
{
  int i, x[6]={0,1,2,3,4,5};
  for(i=0;i<6;i++)
  {
      printf("%d ", x[i]);
  }
  return 0;
}
```

```
#include <stdio.h>
int main()
{
  int i, x[6]={0,1,2,3,4,5};
  for(i=0;i<6;i++)
  {
```

Example

```
#include <stdio.h>
int main()
{
  int i, x[6]={0,1,2,3,4,5};
  for(i=0;i<6;i++)
  {
      printf("%d ", x[i]);
  }
  return 0;
}
```

```
#include <stdio.h>
int main()
{
  int i, x[6]={0,1,2,3,4,5};
  for(i=0;i<6;i++)
  {
      printf("%d ",  *(x+i));
```

Example

```
#include <stdio.h>
int main()
{
  int i, x[6]={0,1,2,3,4,5};
  for(i=0;i<6;i++)
  {
      printf("%d ", x[i]);
  }
  return 0;
}
```

```
#include <stdio.h>
int main()
{
  int i, x[6]={0,1,2,3,4,5};
  for(i=0;i<6;i++)
  {
      printf("%d ",  *(x+i));
  }
```

Example

```
#include <stdio.h>
int main()
{
  int i, x[6]={0,1,2,3,4,5};
  for(i=0;i<6;i++)
  {
      printf("%d ", x[i]);
  }
  return 0;
}
```

```
#include <stdio.h>
int main()
{
  int i, x[6]={0,1,2,3,4,5};
  for(i=0;i<6;i++)
  {
      printf("%d ",  *(x+i));
  }
  return 0;
```

Example

```c
#include <stdio.h>
int main()
{
  int i, x[6]={0,1,2,3,4,5};
  for(i=0;i<6;i++)
  {
      printf("%d ", x[i]);
  }
  return 0;
}
```

```c
#include <stdio.h>
int main()
{
  int i, x[6]={0,1,2,3,4,5};
  for(i=0;i<6;i++)
  {
      printf("%d ",  *(x+i));
  }
  return 0;
}
```

Guess the output?

```
#include <stdio.h>
int main()
{
  int x[5] = {1, 2, 3, 55, 5};
  int *p;
  p=x;
  for(int i=0;i<5;i++)
    printf("%d ", x[i]);

  for(int i=0;i<5;i++)
    printf("%d ", *(p+i));

  for(int i=0;i<5;i++)
    printf("%d ", *p+i);
  return 0;
}
```

Guess the output?

```
#include <stdio.h>
int main()
{
  int x[5] = {1, 2, 3, 55, 5};
  int *p;
  p=x;
  for(int i=0;i<5;i++)
    printf("%d ", x[i]);

  for(int i=0;i<5;i++)
    printf("%d ", *(p+i));

  for(int i=0;i<5;i++)
    printf("%d ", *p+i);
  return 0;
}
```

$\longrightarrow$ 1 2 3 55 5

Guess the output?

```
#include <stdio.h>
int main()
{
  int x[5] = {1, 2, 3, 55, 5};
  int *p;
  p=x;
  for(int i=0;i<5;i++)
    printf("%d ", x[i]);        ⟶  1 2 3 55 5

  for(int i=0;i<5;i++)
    printf("%d ", *(p+i));      ⟶  1 2 3 55 5

  for(int i=0;i<5;i++)
    printf("%d ", *p+i);
  return 0;
}
```

Guess the output?

```
#include <stdio.h>
int main()
{
  int x[5] = {1, 2, 3, 55, 5};
  int *p;
  p=x;
  for(int i=0;i<5;i++)
    printf("%d ", x[i]);         ⟶   1 2 3 55 5

  for(int i=0;i<5;i++)
    printf("%d ", *(p+i));       ⟶   1 2 3 55 5

  for(int i=0;i<5;i++)
    printf("%d ", *p+i);         ⟶   1 2 3 4 5
  return 0;
}
```

Functions

What is Function?

- A function is a block of code that performs a specific task.
- Suppose, a program related to graphics needs to create a circle and color it depending upon the radius and color from the user. You can create two functions to solve this problem:
  - create a circle function
  - color function
- Dividing complex problem into small components makes program easy to understand and use.

Types of Functions

- Depending on whether a function is defined by the user or already included in C compilers, there are two types of functions in C programming
- There are two types of function in C programming:
  - Standard library functions
  - User defined functions

Standard Library Functions

- The standard library functions are built-in functions in C programming to handle tasks such as mathematical computations, I/O processing, string handling etc.
- These functions are defined in the header file. When you include the header file, these functions are available for use.
- For example: ***printf()*** is a standard library function to send formatted output to the screen (display output on the screen).

User defined functions

- As mentioned earlier, C allow programmers to define functions. Such functions created by the user are called user-defined functions.
- You can create as many user-defined functions as you want.

How user-defined function works?

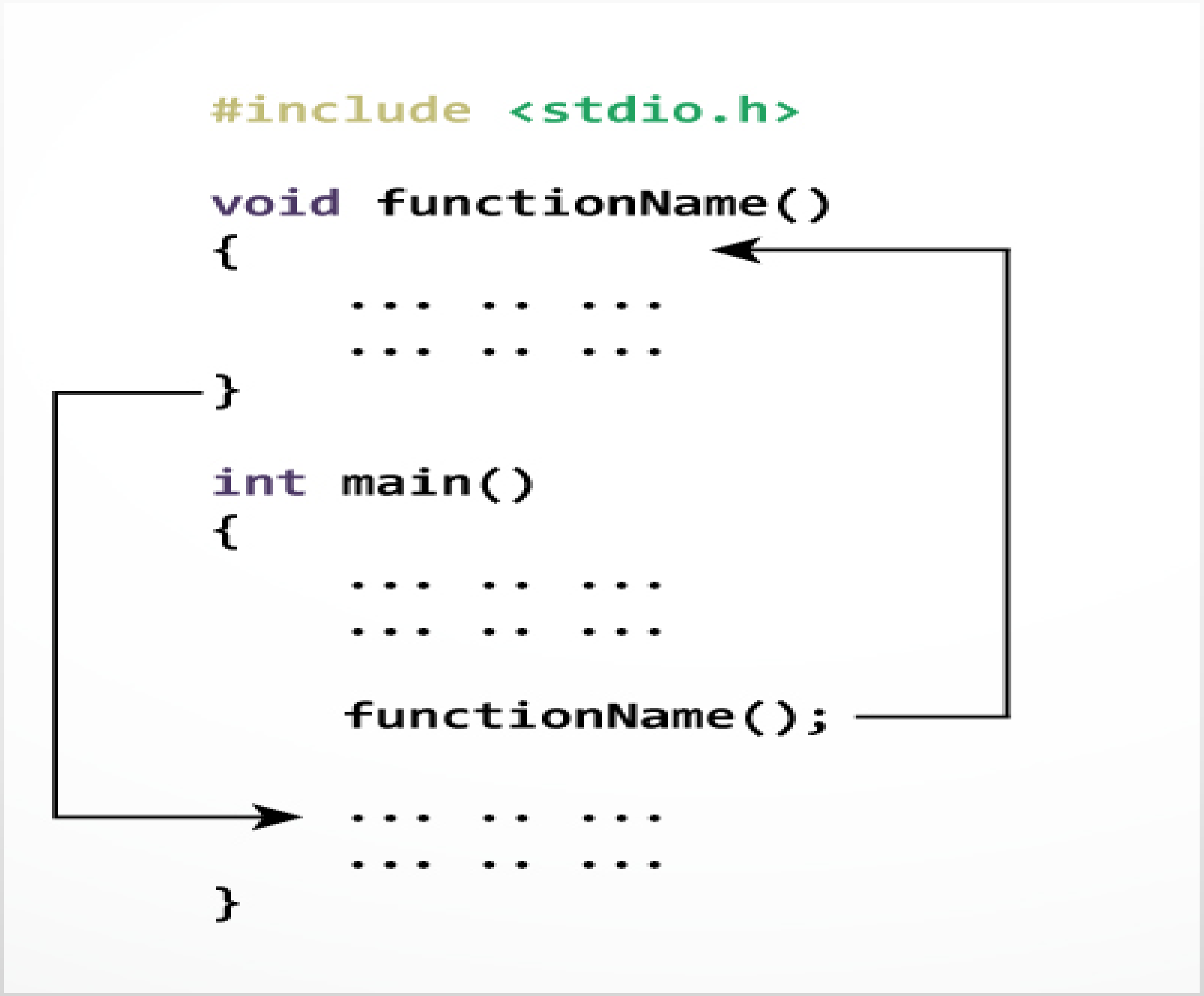
```
#include <stdio.h>
void functionName()
{
    ... .. ...
    ... .. ...
}

int main()
{
    ... .. ...
    ... .. ...

    functionName();

    ... .. ...
    ... .. ...
}
```

How user-defined function works?

- The execution begins from the ***main()*** function.
- When the compiler encounters ***functionname()*** inside the main function, control of the program jumps to

  void ***functionanme()***
- And, the compiler starts executing the codes inside the user-defined function.

How user-defined function works?



```
#include <stdio.h>

void functionName()
{
    ... .. ...
    ... .. ...
}

int main()
{
    ... .. ...
    ... .. ...

    functionName();

    ... .. ...
    ... .. ...
}
```

Advantages of using functions

- The program will be easier to understand, maintain and debug.
- Reusable codes that can be used in other programs
- A large program can be divided into smaller modules. Hence, a large project can be divided among many programmers.

Function prototype

- A function prototype is simply the declaration of a function that specifies function's name, parameters and return type. It doesn't contain function body.
- A function prototype gives information to the compiler that the function may later be used in the program.

```
returnType functionName(type1 argument1, type2 argument2,...);
```

Function prototype

- A function prototype is simply the declaration of a function that specifies function's name, parameters and return type. It doesn't contain function body.
- A function prototype gives information to the compiler that the function may later be used in the program.

```
returnType functionName(type1 argument1, type2 argument2,...);
```

- A prototype declares three attributes associated with function:
  - Its return type
  - The number of its parameters
  - The type of its parameters

Example of a Function

```
#include <stdio.h>

void addNumbers();          // function prototype

int main()
{
    addNumbers();           // function call
    return 0;
}

void addNumbers()           // function definition
{
    int n1,n2;
    printf("Enters two numbers: ");
    scanf("%d %d",&n1,&n2);
    int result;
    result = n1+n2;
    printf("sum = %d",result);
}
```

Returning value from a function

```
#include <stdio.h>

int addNumbers();                       // function prototype

int main()
{
    int sum=addNumbers();               // function call
    printf(“%d”,sum);
    return 0;
}

int addNumbers()                        // function definition
{
    int n1,n2;
    printf("Enters two numbers: ");
    scanf("%d %d",&n1,&n2);
    int result;
    result = n1+n2;
    return result;                      //return statement
}
```

Receiving parameters

```
#include <stdio.h>

int addNumbers(int a, int b);          // function prototype

int main()
{   int n1,n2,sum;
    printf("Enters two numbers: ");
    scanf("%d %d",&n1,&n2);
    sum = addNumbers(n1, n2);          // function call
    printf("sum = %d",sum);
    return 0;
}

int addNumbers(int a,int b)            // function definition
{
    int result;
    result = a+b;
    return result;                     // return statement
}
```

Receiving parameters

```
#include <stdio.h>

int addNumbers(int a, int b);          // function prototype

int main()
{   int n1,n2,sum;
    printf("Enters two numbers: ");
    scanf("%d %d",&n1,&n2);
    sum = addNumbers(n1, n2);          // function call
    printf("sum = %d",sum);
    return 0;
}

int addNumbers(int a,int b)            // function definition
{
    int result;
    result = a+b;
    return result;                     // return statement
}
```

Receiving parameters

```
#include <stdio.h>

int addNumbers(int a, int b);          // function prototype

int main()
{   int n1,n2,sum;
    printf("Enters two numbers: ");
    scanf("%d %d",&n1,&n2);
    sum = addNumbers(n1, n2);          // function call
    printf("sum = %d",sum);
    return 0;
}

int addNumbers(int a,int b)            // function definition
{
    int result;
    result = a+b;
    return result;                     // return statement
}
```

Example

- The function prototype is not needed if the user-defined function is defined before the main() function.

Example

- The function prototype is not needed if the user-defined function is defined before the main() function.

```
#include <stdio.h>
int addNumbers(int a,int b)         // function definition
{
    int result;
    result = a+b;
    return result;                  // return statement
}

int main()
{   int n1,n2,sum;
    printf("Enters two numbers: ");
    scanf("%d %d",&n1,&n2);
    sum = addNumbers(n1, n2);       // function call
    printf("sum = %d",sum);
    return 0;
}
```