CSE1062 | CCS1063 ‘Practicals’ {

[Fundamentals of Computer Programming]

< Tutorial Session 9 - Pointers >

}

What is an Pointer?

Pointers (pointer variables) are special variables that are used to store addresses rather than values.

```
int* p;
int *p1;
int * p2;
int* p1, p2;
```

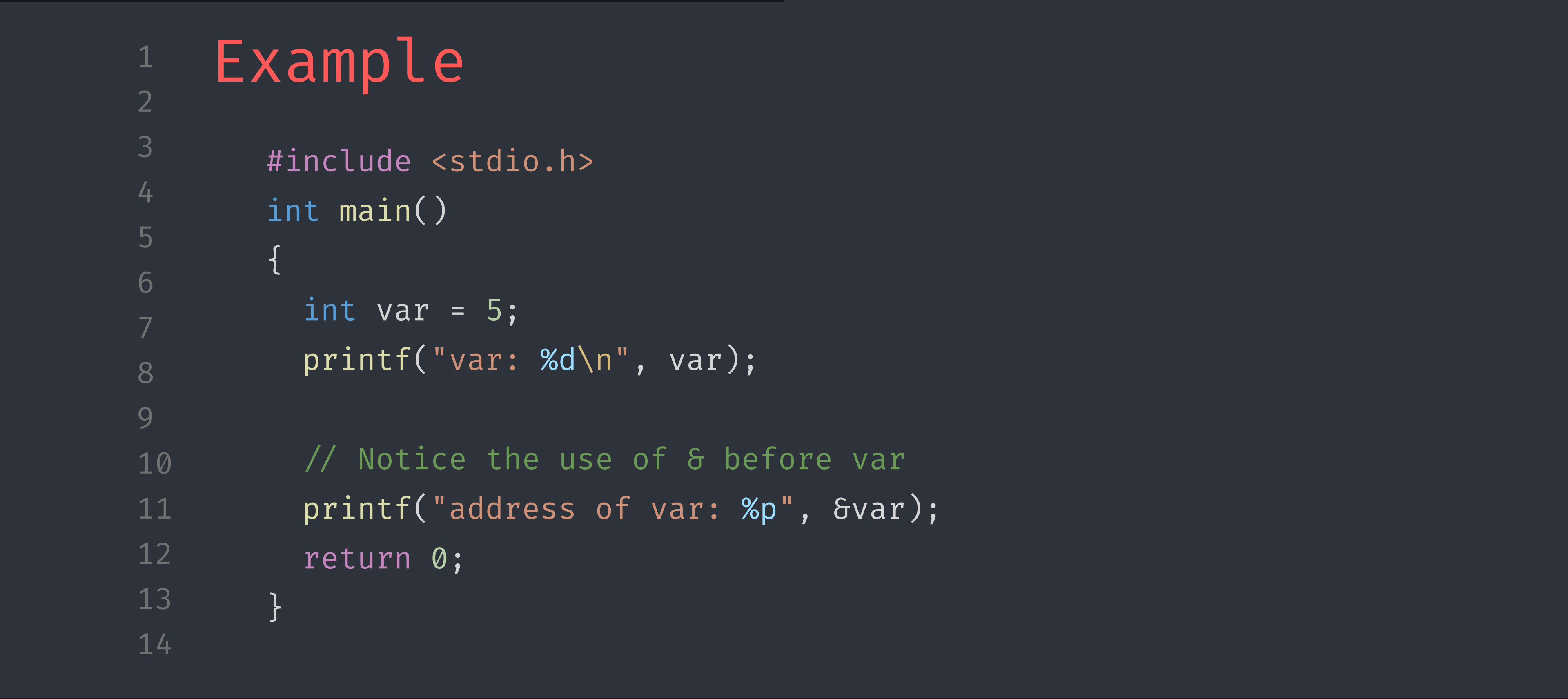
https://www.programiz.com/c-programming/c-pointers

Address in C

If you have a variable var in your program, &var will give you its address in the memory.

We have used address numerous times while using the scanf() function.

https://www.programiz.com/c-programming/c-pointers



Example

```c
#include <stdio.h>
int main()
{
  int var = 5;
  printf("var: %d\n", var);

  // Notice the use of & before var
  printf("address of var: %p", &var);
  return 0;
}
```

https://www.programiz.com/c-programming/c-pointers

Assigning addresses to Pointers

```c
int* pc, c;
c = 5;
pc = &c;
```

Here, 5 is assigned to the c variable. And, the address of c is assigned to the pc pointer.

https://www.programiz.com/c-programming/c-pointers

Get Value of Thing Pointed by Pointers

```c
int* pc, c;
c = 5;
pc = &c;
printf("%d", *pc);   // Output: 5
```

Here, the address of c is assigned to the pc pointer. To get the value stored in that address, we used *pc.

In the above example, pc is a pointer, not *pc. You cannot and should not do something like *pc = &c;

By the way, * is called the dereference operator (when working with pointers). It operates on a pointer and gives the value stored in that pointer.

https://www.programiz.com/c-programming/c-pointers

Changing Value Pointed by Pointers

```
int* pc, c;
c = 5;
pc = &c;
c = 1;
printf("%d", c);    // Output: 1
printf("%d", *pc);  // Ouptut: 1
```

https://www.programiz.com/c-programming/c-pointers

Example

```c
int* pc, c;
c = 5;
pc = &c;
*pc = 1;
printf("%d", *pc);   // Ouptut: ?
printf("%d", c);     // Output: ?
```

https://www.programiz.com/c-programming/c-pointers

Example: Working of Pointers

```
Address of c: 2686784
Value of c: 22

Address of pointer pc:
2686784
Content of pointer pc: 22

Address of pointer pc:
2686784
Content of pointer pc: 11

Address of c: 2686784
Value of c: 2
```

https://www.programiz.com/c-programming/c-pointers

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

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

   pc = &c;
   printf("Address of pointer pc: %p\n",
pc);
   printf("Content of pointer pc: %d\n\n",
*pc); // 22

   c = 11;
   printf("Address of pointer pc: %p\n",
pc);
   printf("Content of pointer pc: %d\n\n",
*pc); // 11

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

Common mistakes when working with pointers

```
int c, *pc;

// pc is address but c is not
pc = c;  // Error

// &c is address but *pc is not
*pc = &c;  // Error

// both &c and pc are addresses
pc = &c;  // Not an error

// both c and *pc are values
*pc = c;  // Not an error
```

https://www.programiz.com/c-programming/c-pointers

Common mistakes when working with pointers

```c
#include <stdio.h>
int main() {
    int c = 5;
    int *p = &c;

    printf("%d", *p);  // 5
    return 0;
}
```

https://www.programiz.com/c-programming/c-pointers

Arrays and Pointers

An array is a block of sequential data. Let's write a program to print addresses of array elements.

https://www.programiz.com/c-programming/c-pointers

```c
#include <stdio.h>
int main() {
   int x[4];
   int i;

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

   printf("Address of array x: %p", x);

   return 0;
 }
```

Example

```
Output:

Enter 6 numbers: 2

3

4

4

12

4

Sum = 29
```

https://www.programiz.com/c-programming/c-pointers

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

  int i, x[6], sum = 0;

  printf("Enter 6 numbers: ");

  for(i = 0; i < 6; ++i) {
  // Equivalent to scanf("%d", &x[i]);
      scanf("%d", x+i);

  // Equivalent to sum += x[i]
      sum += *(x+i);
  }

  printf("Sum = %d", sum);

  return 0;
  }
```

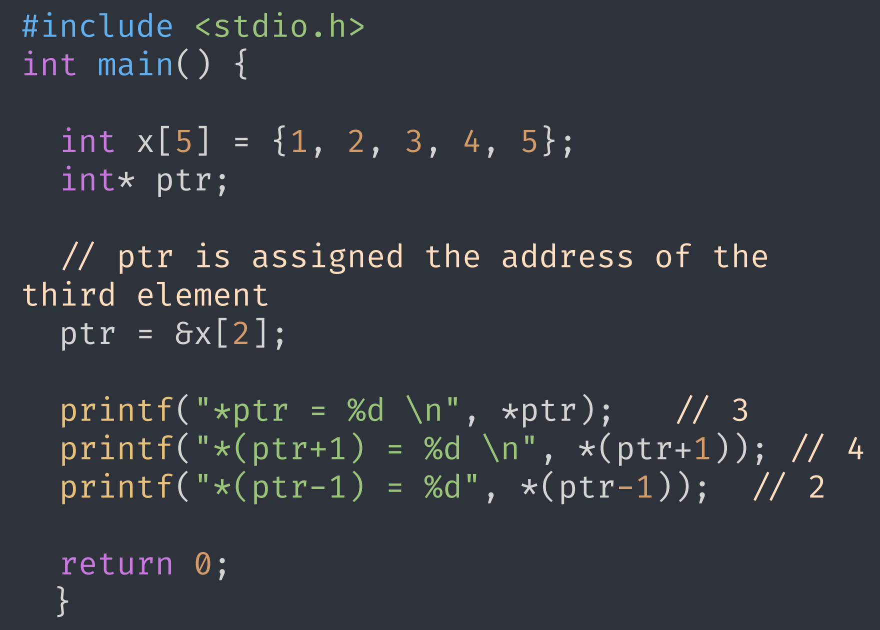
Example

```
Output:

*ptr = 3

*(ptr+1) = 4

*(ptr-1) = 2
```



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

  int x[5] = {1, 2, 3, 4, 5};
  int* ptr;

  // ptr is assigned the address of the
third element
  ptr = &x[2];

  printf("*ptr = %d \n", *ptr);   // 3
  printf("*(ptr+1) = %d \n", *(ptr+1)); // 4
  printf("*(ptr-1) = %d", *(ptr-1));  // 2

  return 0;
  }
```

https://www.programiz.com/c-programming/c-pointers

C Pass Addresses and Pointers

In C programming, it is also possible to pass addresses as arguments to functions.

To accept these addresses in the function definition, we can use pointers. It's because pointers are used to store addresses.

https://www.programiz.com/c-programming/c-pointers

Example: Pass Addresses to Functions

output will be:

```
num1 = 10

num2 = 5
```

```c
#include <stdio.h>
void swap(int *n1, int *n2);

int main()
{
    int num1 = 5, num2 = 10;
    // address of num1 and num2 is passed
    swap( &num1, &num2);
    printf("num1 = %d\n", num1);
    printf("num2 = %d", num2);
    return 0;
}
void swap(int* n1, int* n2)
{
    int temp;
    temp = *n1;
    *n1 = *n2;
    *n2 = temp;
  }
```

https://www.programiz.com/c-programming/c-pointers

Example: Passing Pointers to Functions

Here, the value stored at p, *p, is 10 initially.

We then passed the pointer p to the addOne() function. The ptr pointer gets this address in the addOne() function.

Inside the function, we increased the value stored at ptr by 1 using (*ptr)++;. Since ptr and p pointers both have the same address, *p inside main() is also 11.

```c
#include <stdio.h>

void addOne(int* ptr) {
  (*ptr)++; // adding 1 to *ptr
}

int main()
{
  int* p, i = 10;
  p = &i;
  addOne(p);

  printf("%d", *p); // 11
  return 0;
  }
```

https://www.programiz.com/c-programming/c-pointers

C Dynamic Memory Allocation

Array is a collection of a fixed number of values. Once the size of an array is declared, you cannot change it.

Sometimes the size of the array you declared may be insufficient. To solve this issue, you can allocate memory manually during run-time. This is known as dynamic memory allocation in C programming.

To allocate memory dynamically, library functions are malloc(), calloc(), realloc() and free() are used. These functions are defined in the <stdlib.h> header file.

https://www.programiz.com/c-programming/c-pointers

malloc()

The name "malloc" stands for memory allocation.

The malloc() function reserves a block of memory of the specified number of bytes. And, it returns a pointer of void which can be casted into pointers of any form.

```
ptr = (castType*) malloc(size);
```

https://www.programiz.com/c-programming/c-pointers

Example

```c
ptr = (float*) malloc(100 * sizeof(float));
```

The above statement allocates 400 bytes of memory. It's because the size of float is 4 bytes. And, the pointer ptr holds the address of the first byte in the allocated memory.

The expression results in a NULL pointer if the memory cannot be allocated.

https://www.programiz.com/c-programming/c-pointers

calloc()

The name "calloc" stands for contiguous allocation.

The malloc() function allocates memory and leaves the memory uninitialized, whereas the calloc() function allocates memory and initializes all bits to zero.

```
ptr = (castType*)calloc(n, size);
```

https://www.programiz.com/c-programming/c-pointers

Example

```c
ptr = (float*) calloc(25, sizeof(float));
```

The above statement allocates contiguous space in memory for 25 elements of type float.

https://www.programiz.com/c-programming/c-pointers

free()

Dynamically allocated memory created with either calloc() or malloc() doesn't get freed on their own. You must explicitly use free() to release the space.

```
free(ptr);
```

This statement frees the space allocated in the memory pointed by ptr.

https://www.programiz.com/c-programming/c-pointers

Example: malloc() and free()

```
Enter number of elements: 3

Enter elements: 100

20

36

Sum = 156
```

https://www.programiz.com/c-programming/c-pointers

```c
// Program to calculate the sum of n numbers
entered by the user

#include <stdio.h>
#include <stdlib.h>
int main() {
  int n, i, *ptr, sum = 0;
  printf("Enter number of elements: ");
  scanf("%d", &n);
  ptr = (int*) malloc(n * sizeof(int));
  // if memory cannot be allocated
  if(ptr == NULL) {
    printf("Error! memory not allocated.");
    exit(0);
  }
  printf("Enter elements: ");
  for(i = 0; i < n; ++i) {
    scanf("%d", ptr + i);
    sum += *(ptr + i);
  }
  printf("Sum = %d", sum);
  // deallocating the memory
  free(ptr);
  return 0;
  }
```

Example: calloc() and free()

```
Enter number of elements: 3

Enter elements: 100

20

36

Sum = 156
```

https://www.programiz.com/c-programming/c-pointers

```c
// Program to calculate the sum of n numbers
entered by the user

#include <stdio.h>
#include <stdlib.h>

int main() {
  int n, i, *ptr, sum = 0;
  printf("Enter number of elements: ");
  scanf("%d", &n);

  ptr = (int*) calloc(n, sizeof(int));
  if(ptr == NULL) {
    printf("Error! memory not allocated.");
    exit(0);
  }

  printf("Enter elements: ");
  for(i = 0; i < n; ++i) {
    scanf("%d", ptr + i);
    sum += *(ptr + i);
  }

  printf("Sum = %d", sum);
  free(ptr);
  return 0;
 }
```

realloc()

If the dynamically allocated memory is insufficient or more than required, you can change the size of previously allocated memory using the realloc() function.

```
ptr = realloc(ptr, x);
```

here, ptr is reallocated with a new size x.

https://www.programiz.com/c-programming/c-pointers

Example

```
Enter size: 2

Addresses of previously
allocated memory:

26855472

26855476


Enter the new size: 4

Addresses of newly allocated
memory:

26855472

26855476

26855480
```

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
  int *ptr, i , n1, n2;
  printf("Enter size: ");
  scanf("%d", &n1);

  ptr = (int*) malloc(n1 * sizeof(int));

  printf("Addresses of previously allocated
memory:\n");
  for(i = 0; i < n1; ++i)
    printf("%pc\n",ptr + i);

  printf("\nEnter the new size: ");
  scanf("%d", &n2);
  // rellocating the memory
  ptr = realloc(ptr, n2 * sizeof(int));
  printf("Addresses of newly allocated
memory:\n");
  for(i = 0; i < n2; ++i)
    printf("%pc\n", ptr + i);

  free(ptr);

  return 0;
  }
```

Pointers Arithmetic

- Arithmetic operators work as usual on ordinary data types.
  - `int a = 1; a++ // a == 2`
- It gets a bit complicated when arithmetic operators are used on pointers.
  - `int *p = 0x8004; p++;`
- Compiler knows that p is a pointer to integer type data, so an increment to it should point to next integer in memory.
  - Hence 0x8008.

Pointers Arithmetic

So an arithmetic operator increase or decrease its contents by the size of data type it points to.

```
int *pi = 0x8004;
double *pd = 0x9004;
char *pc = 0xa004;
pi++; // pi == 0x8008 (size of int = 4 byte)
pd++; // pd == 0x900C (size of double = 8 byte)
pc++; // pc == 0xa005 (size of char = 1 byte)
```

Only ‘+’ and ‘-’ operators are allowed. ‘*’ and ‘/’ are meaningless.

Pointer to Pointer

Pointer variable is just a place-holder of an address value, and itself is a variable.

Hence a pointer can hold address of other pointer variable. In that case it is called a “double pointer”

```
int *p;
int **pp;
pp = &p
```

Pointer to Pointer

E.g. A function may like to return a pointer value.

```
void pp_example (int **p){
*p = 0x8004;
}
int *p; pp_example (&p);
```

Pointer Pitfalls

* Since pointer holds address of memory location, it must never be used without proper initialization.

* An uninitialized pointer may hold address of some memory location that is protected by Operating system.

* In such case, de-referencing a pointer may crash the program.

* An initialized pointer does not know the memory location, it is pointing to is, holds a valid value or some garbage.

* A pointer cannot track boundaries of an array.

```
Thanks; {

    ‘Do you have any questions?’

        < bgamage@sjp.ac.lk >

}
```