The C Programming Language

Strings, Pointers, Dynamic Memory Allocations

ITF225191 20 H

Høgskolen i Østfold

Contents

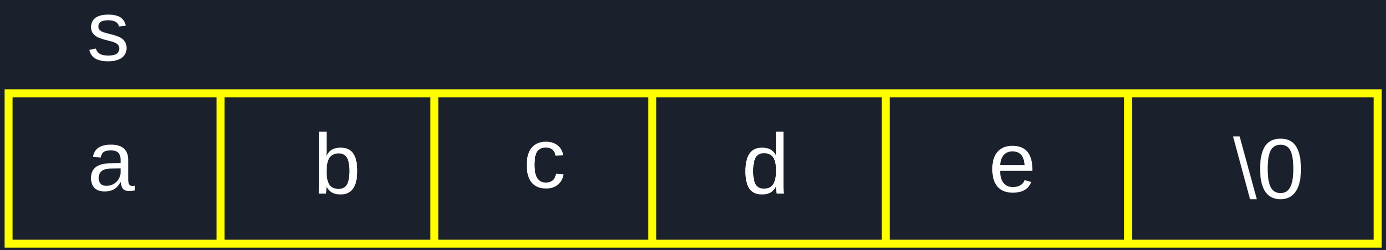
Contents

Strings

- One-dimensional arrays of type char
- Terminated with ‘\0’ or NULL
    - Byte with all bits off
- Strings can be manipulated in the same way with arrays

Example

- char s[ ] = "abcde";
- char s[ ] = {'a', 'b', 'c', 'd', 'e', `\0’};



- Here, s[0] = a, s[1] = b, s[2] = c, s[3] = d, s[4] = e, and s[5] = NULL.

Library functions of string

- C provides many string handling functions in standard library with header string.h

- strcat(): concatenates (joins) two strings
- strcomp(): compares two strings character by character
- strcpy(): copies the string pointed by source (including the null character) to the destination
- strlen(): calculates the length of a given string

Contents

Introduction

- A variable in a program is stored in a certain number of bytes at a particular memory location (address)
- Pointers: used to access memory and manipulate address
- If v is a variable, then &v gives its memory address
- &: unary operator

Declarations

- Pointers can be declared in programs and then used to take address value
- int *p;
  - p: type of pointer to int
  - Its value range includes a special address 0 and set of positive integers that present machine addresses
- Example
  - p = 0;
  - p = NULL; //same as p = 0;
  - p = &i; // pointing to integer i;
  - p = (int *) 1776; // absolute address

Characteristic

- If p is a pointer
    - * p : value of the variable at address p

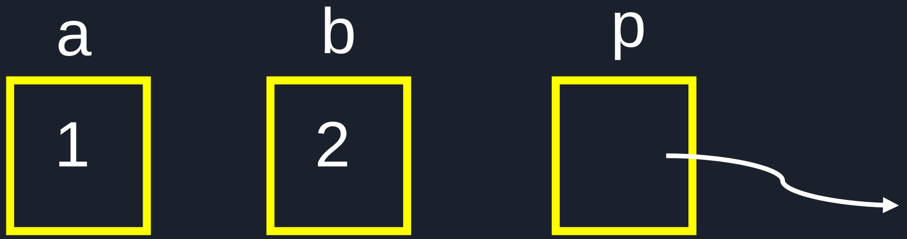
- Direct value of p : address of memory location

- Indirect value of p : value stored at address p

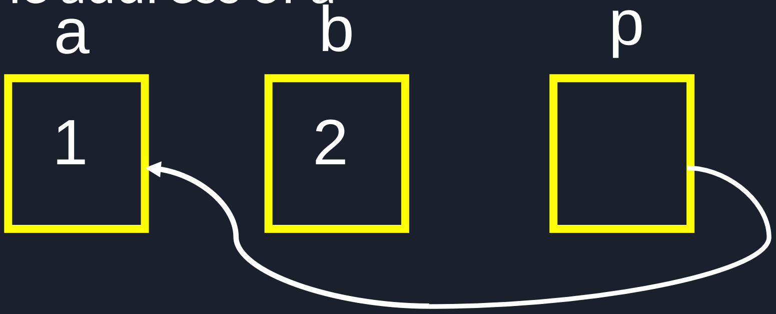
- *: inverse operator of &

Example

- Int a = 1, b = 2, *p



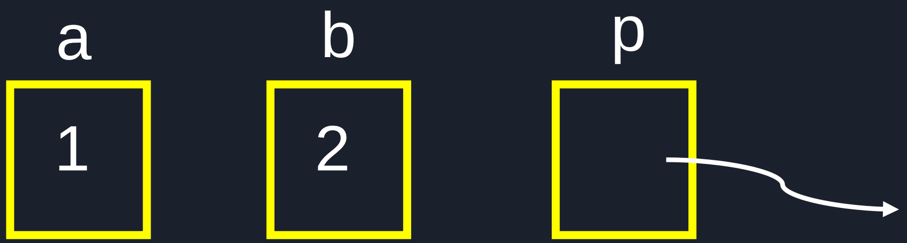
- Think of the pointer as an arrow, but it is not yet assigned a value
- p = &a: p is assigned the address of a



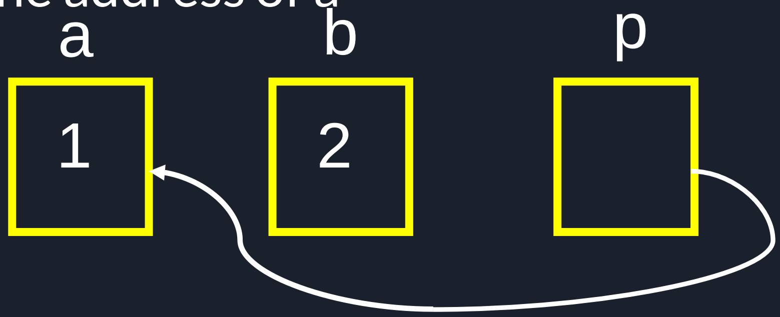
- b= *p; b is assigned the value pointed to by p
- b=?

Example

- Int a = 1, b = 2, *p



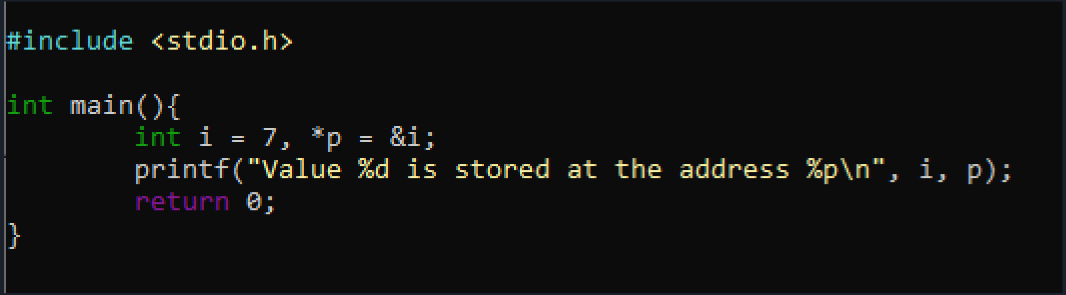
- Think of the pointer as an arrow, but it is not yet assigned a value
- p = &a: p is assigned the address of a



- b= *p; b is assigned the value pointed to by p
- b=? (b=a=1)

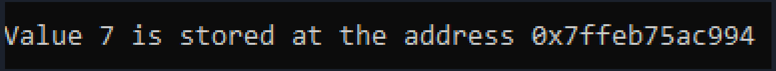
Characteristic

- A pointer can be initialized in a declaration
- The variable p is of type int and its initial value is &i.
- The declaration of i must occur before we take its address



```c
#include <stdio.h>

int main(){
        int i = 7, *p = &i;
        printf("Value %d is stored at the address %p\n", i, p);
        return 0;
}
```

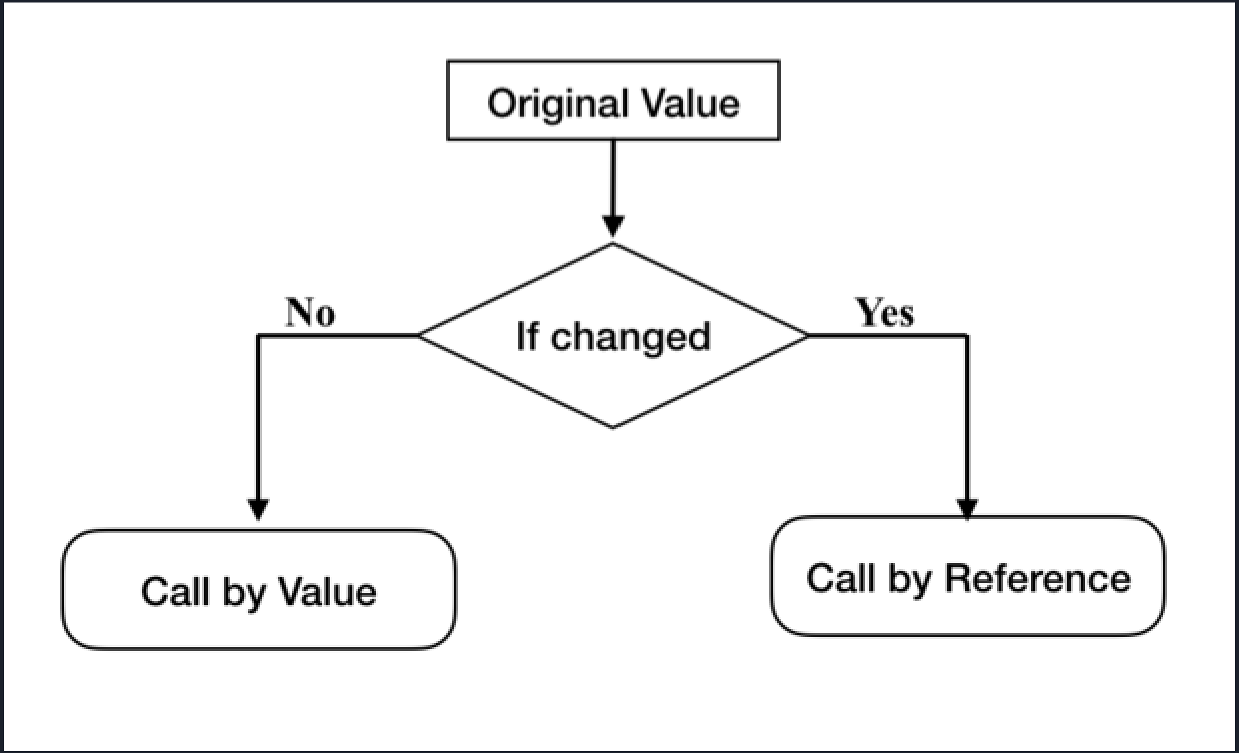


```
Value 7 is stored at the address 0x7ffeb75ac994
```

Call-by-reference

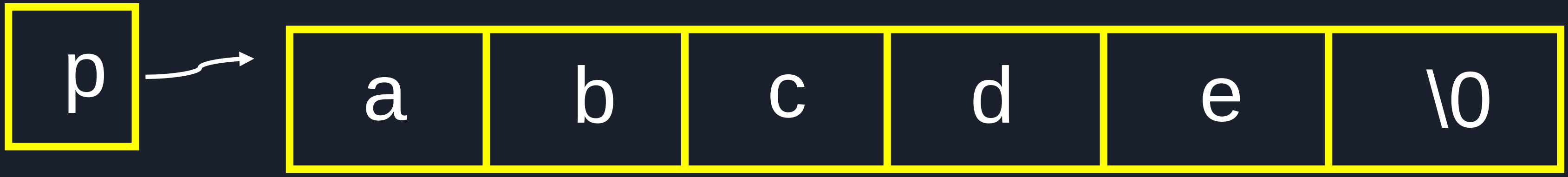
- Call-by-value:
  - When variables are passed as arguments to a function, their values are copied into corresponding parameters in the functions
  - Variables are not changed in the calling environments
- Call-by-reference
  - Passing address (reference) of variables
  - When function is called, variables are changed in the calling environment

Call-by-reference

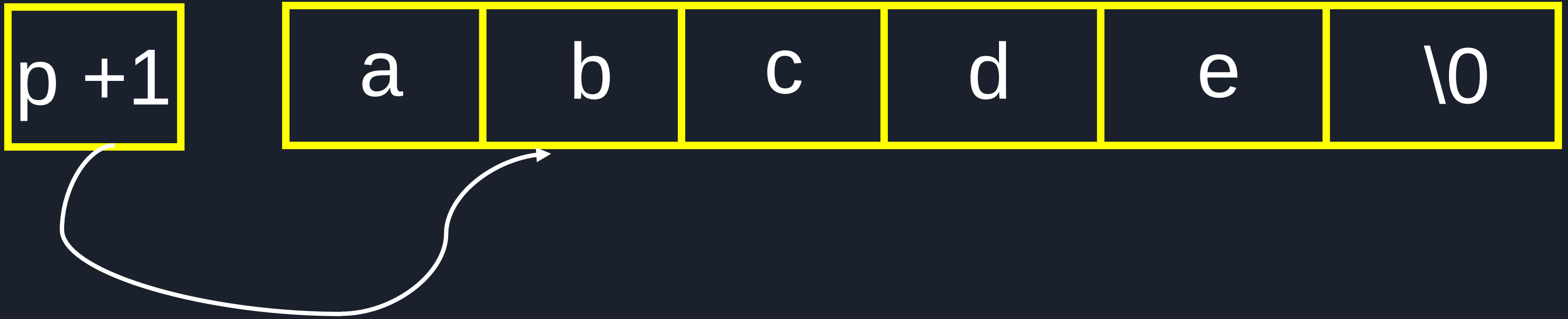


Pointers and Strings

- char *p= "abcde";

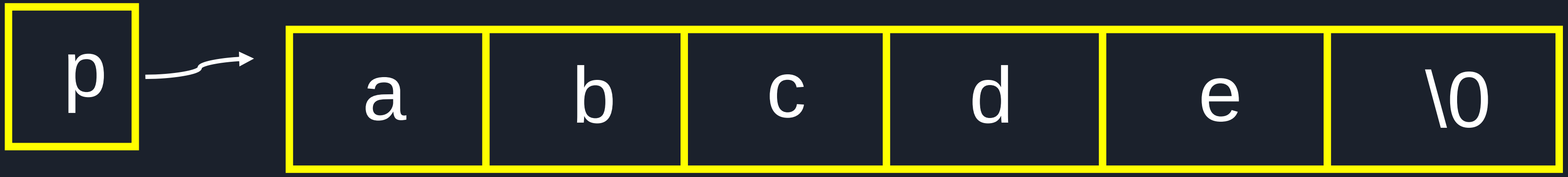


- p+1

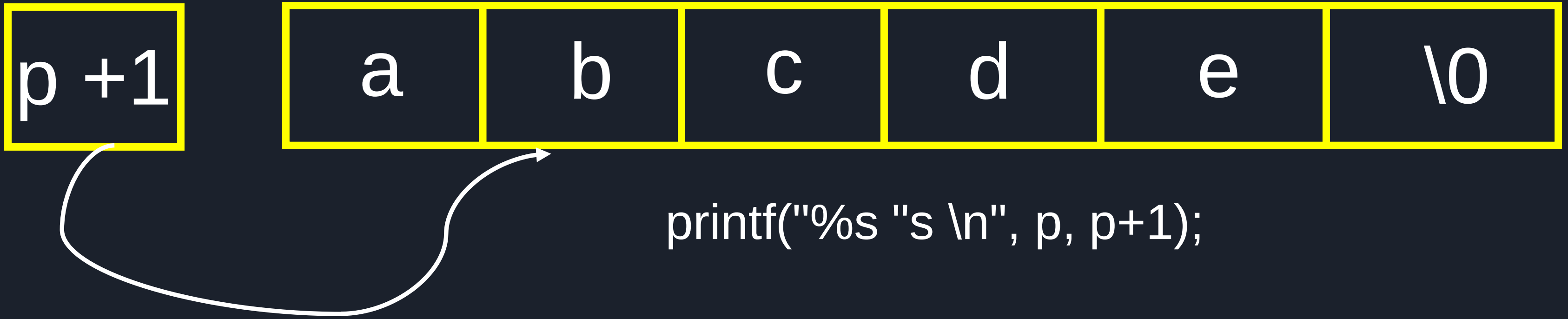


Pointers and Strings

- char *p= "abcde";



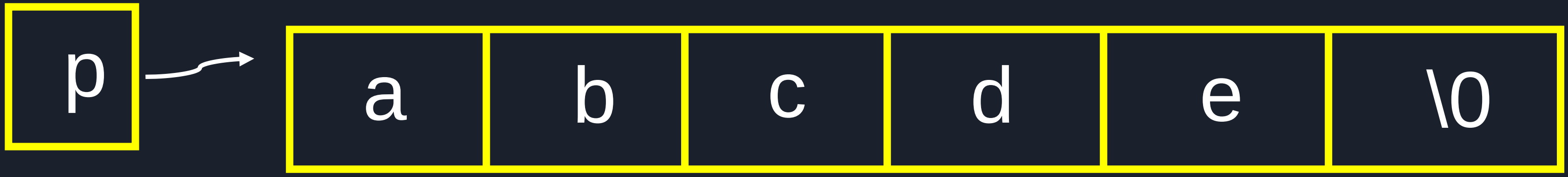
- p+1



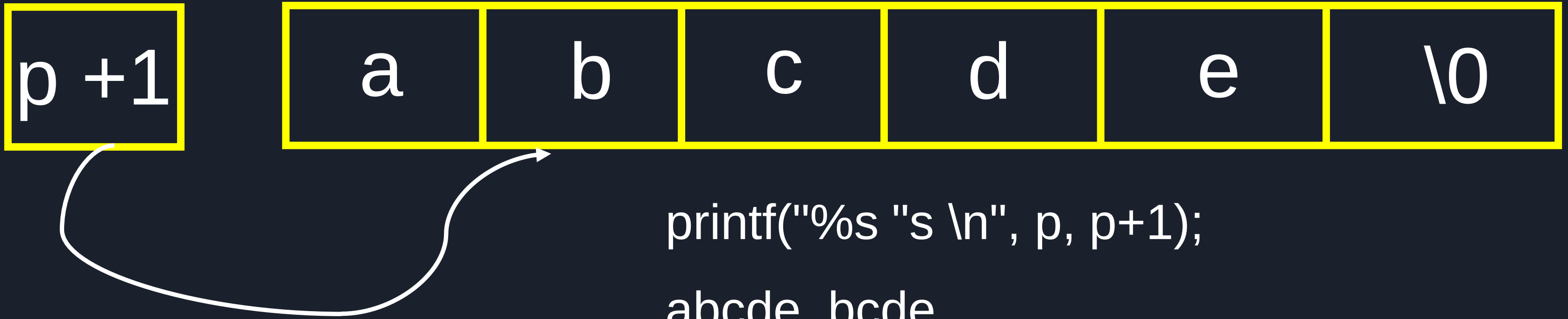
printf("%s "s \n", p, p+1);

Pointers and Strings

- char *p= "abcde";



- p+1



```
printf("%s "s \n", p, p+1);
abcde  bcde
```

Pointers and Arrays

- A pointer variable can take different address as value
- An array name is an address, or pointer, that is fixed
- The following are illegal
- a=p;
- ++a;
- a+ = 2
- &a

Pointers and Arrays

- Suppose that A is an array and that i is an int, then the following expressions are the same: A[i] and *(A+i).

- If p is a pointer then

  p = A equivalent to p = &A[0];
  p = A + 1 equivalent to p = &A[1];

Pointers and Arrays

- Same expression:

```
for (p = a; p < &a[N]; ++p)
        sum += *p;
```

```
for (i = 0; i < N; ++i)
            sum += *(a+i);
```

```
p=a;
for(i = 0; i < N; ++i)
      sum += p[i];
```

Contents

Dynamic Memory Allocation

- C provides two functions in stdlib.h
  - calloc( ): contiguous memory allocation
  - malloc( ): memory allocation

- calloc( ) and malloc( ): crate space for arrays, structures, and unions.

calloc( )

```
# stdlib.h
 int *a;
 int n;
 scanf(“%d”,&n);
 a = calloc(n,sizeof(int));
```

- Allocate contiguous space in memory for an array of n elements
- The space is initialized with all bets set to zero

malloc( )

```
# stdlib.h
 int *a;
 int n;
 scanf("%d",&n);
 a = malloc(n*sizeof(int));
```

- Does not initialize the memory allocations
- Faster than calloc( )

free( )

- Programmer must use free() to free the allocated memory
- free(a)

Example

- Memory allocation for one-dimensional array

```
int *A;
A = (int*)malloc(n*sizeof(int));
if (A== NULL){
        printf(“Memory Allocation error!\n”);
        exit(0);
}
////
///
free(A);
```

Example

- Memory allocation for two-dimensional array with m rows, n columns

```
int **A;
A = (int**)malloc(m*sizeof(int));
for (int I = 0; i< m; i++){
        A[i] = (int*) malloc(n*sizeof(int));
}
////
///
free(A[i]);
free(A);
```

Contents

int main(int argc, char *argv[ ])

- Two arguments named argc and argv can be used with main()to communicate with the OS
- Int main(int argc, char *argv[])
- argc provides a count of the number of command line arguments
- Array argv is an array of pointers that are the words that make up the command line. Because the element argv [0] contains the name of the command itself, the value of argc is at least 1.

References

- A book on C, Al Kelley and Ira Pohl, 4th Edition.