Arrays and Pointer Arithmetic

Arrays

```
1  int main () {
2
3      int  var[] = {10, 100, 200};
4      for(int i=0; i<=2; i++) {
5          printf("%d ", var[i]);
6      }
7  }
```

Arrays with Pointers

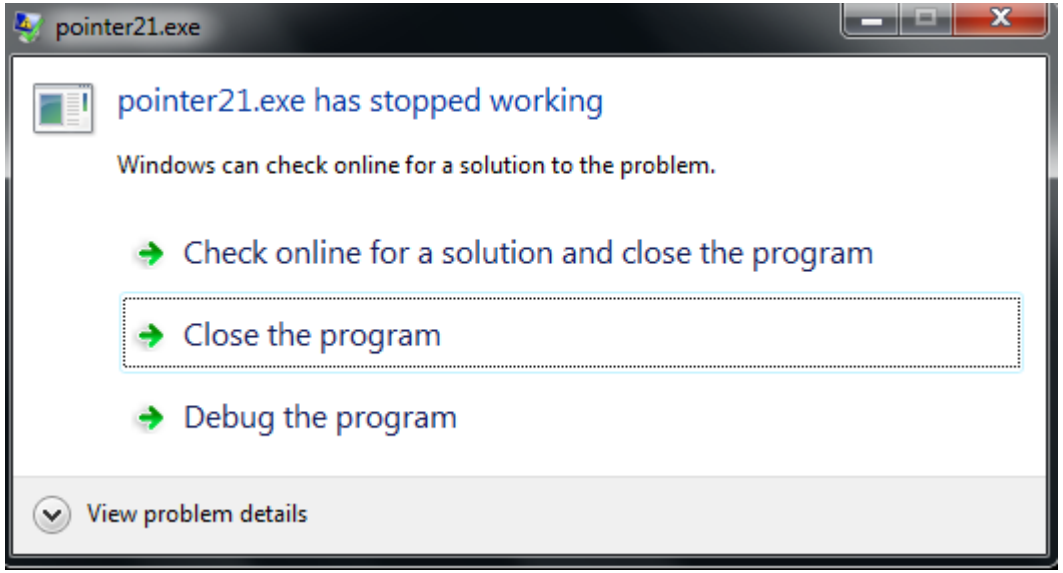
```
1   int main () {
2
3       int  var[] = {10, 100, 200};
4       int *ptr;
5       ptr = var;
6       for(int i=0; i<=2; i++) {
7             printf("%d ", *ptr);
8             ptr++;
9       }
10  }
```

```
10 100 200
Process returned 0 (0x0)   execution time : 0.390 s
Press any key to continue.
```

Almost Everything Can be Done Using Pointers

Why did this crash?

```c
int main()
{
    int foo = 0;
    scanf("%d", foo);
    printf("%d", foo);
}
```



We Fixed It!

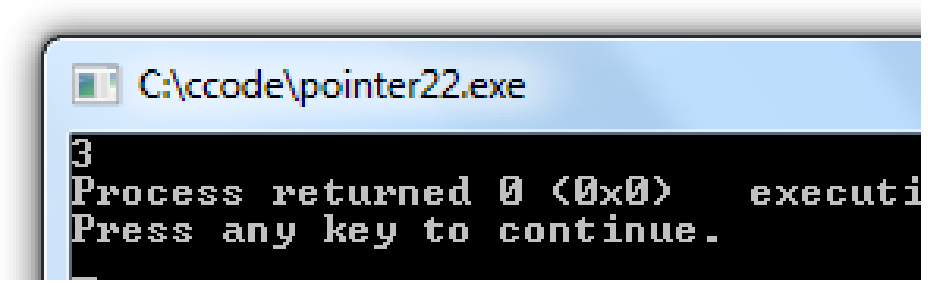
```
1  int main()
2  {
3      int foo = 0;
4      scanf("%d", &foo);
5      printf("%d", foo);
6  }
```

Segmentation Fault

- Accessed an invalid location (where program has no access)
  - Either reading or writing
- Segments = Memory chunks
  - Only some segments are available for programs to access
- If your app is crashing due to segmentation fault, you may be
  - Accessing beyond the array boundary
  - Poor pointer arithmetic pushes the address to invalid locations
  - You just missed & or * somewhere!

Dynamic Memory Allocation

```c
int main() {
    int * arr = malloc(3 * sizeof(int));
    arr[0] = 1;
    arr[1] = 2;
    arr[2] = 3;
    printf("%d", arr[2]);
    free(arr);
}
```



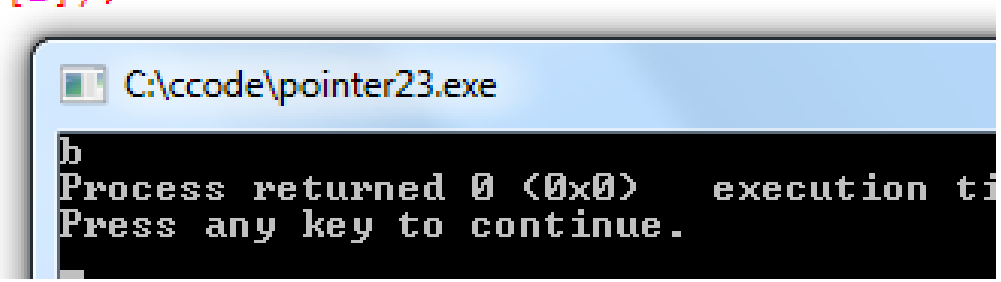
Oh! Is this allowed?

```
1  int main() {
2      int * arr = malloc(6);
3      arr[0] = 1;
4      arr[1] = 'a';
5      arr[2] = 'b';
6      printf("%c", arr[2]);
7      free(arr);
8  }
```

You can do anything with pointers!!

- Do not use the address after a call to free()
  - We call them **Dangling Pointers**!

```
1  int main() {
2      int * arr = malloc(6);
3      arr[0] = 1;
4      arr[1] = 'a';
5      arr[2] = 'b';
6      printf("%c", arr[2]);
7      free(arr);
8  }
9
```

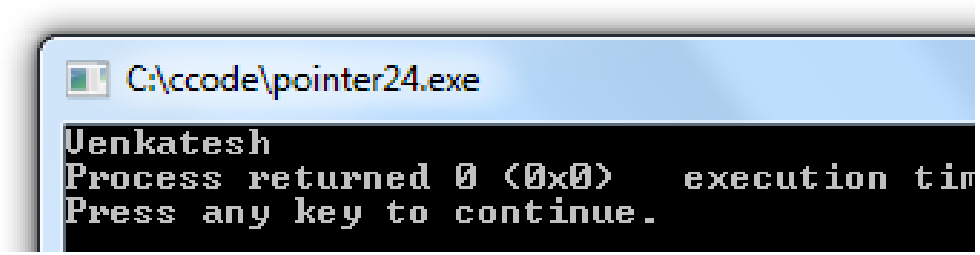


Pointers and Strings

Revisiting Strings

- Declare, Initialize and Print Strings

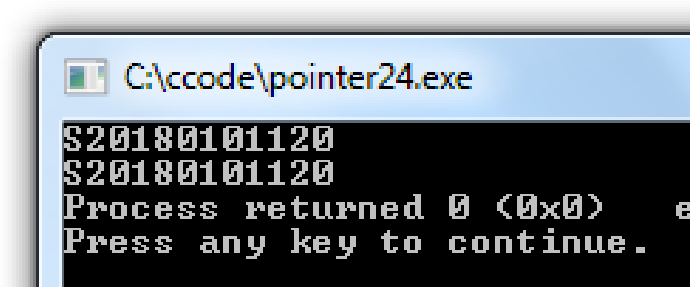
```c
int main()
{
    // declare and initialize string
    char str[] = "Venkatesh";

    // print string
    printf("%s",str);

    return 0;
}
```



What is the output?

- If input is your roll number

```
1  int main()
2  {
3      // declare and initialize string
4      char str[12];
5      scanf("%s", str);
6      printf("%s",str);
7      return 0;
8  }
9
10
```



Length of a String

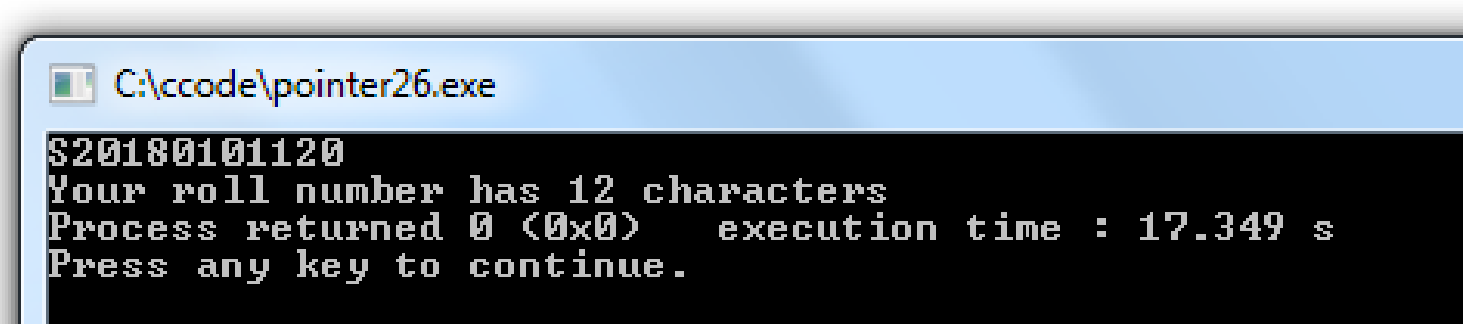
- strlen prints the number of characters.
- Note: str is address of the array. So, &str is not required in scanf.

```c
int main()
{
    // declare and initialize string
    char str[12];
    scanf("%s", str);
    printf("Your roll number has %d characters",strlen(str));
    return 0;
}
```

```
C:\ccode\pointer25.exe
S20180101120
Your roll number has 12 characters
Process returned 0 (0x0)   execution time : 8.894 s
Press any key to continue.
```

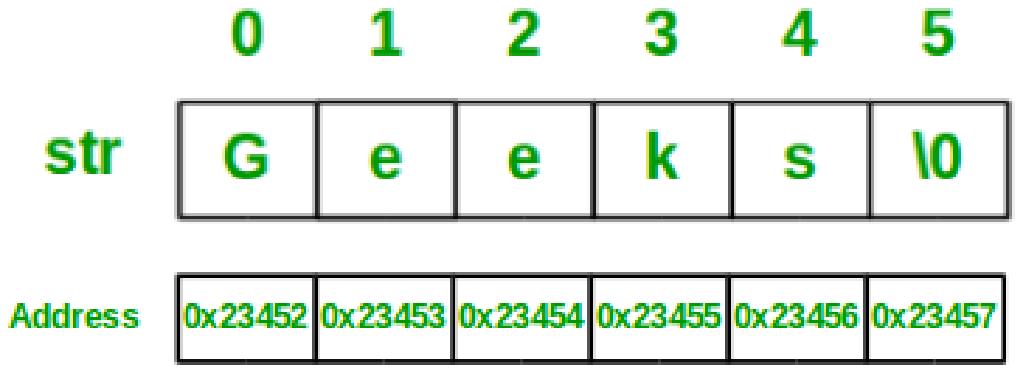
Strings Using Pointers

```c
int main()
{
    // declare and initialize string
    char *str;
    scanf("%s", str);
    printf("Your roll number has %d characters",strlen(str));
    return 0;
}
```



Arrays and Pointers

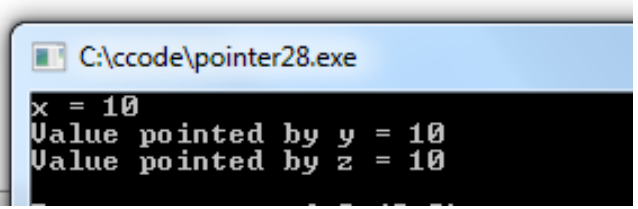
- char str[] = “Geeks”;
- char *str = "Geeks";



Source: https://www.geeksforgeeks.org/strings-in-c-2/

Pointer to a Pointer

```c
int main () {

   int  x;
   int  *y;
   int  **z;

   x = 10;
   y = &x;
   z = &y;

   printf("x = %d\n", x);
   printf("Value pointed by y = %d\n", *y );
   printf("Value pointed by z = %d\n", **z);

   return 0;
}
```



Summary

- Many C features use pointers behind the scene.
- Pointer-based code can be complicated.
  - Addition of two pointers is not allowed.
  - Subtraction is allowed in arrays to get the offset.
- Avoid using pointers unnecessarily.