ESc 101: Fundamentals of Computing

Lecture 21

Feb 18, 2010

Outline

1 Pointers

2 SCANF

Pointer Arithmetic

- Since pointer variables store memory addresses, we can add and subtract from them to access other addresses!
- For a pointer variable `y`, `*(y+1)` refers to the next memory location.
- Depending on the type of variable, this can be one or more bytes away.
- Caution: This must be done with extreme care!!
- If we shift the pointer to a location outside the designated memory locations then unpredictable things may happen.

Pointer Arithmetic

- Since pointer variables store memory addresses, we can add and subtract from them to access other addresses!
- For a pointer variable `y`, `*(y+1)` refers to the next memory location.
- Depending on the type of variable, this can be one or more bytes away.
- Caution: This must be done with extreme care!!
- If we shift the pointer to a location outside the designated memory locations then unpredictable things may happen.

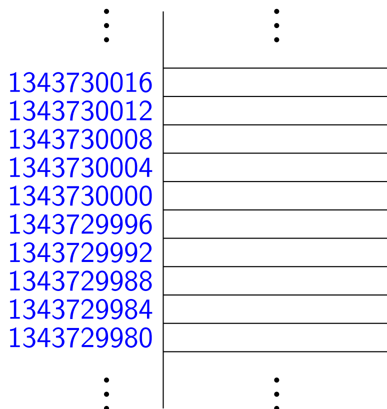
Pointer Arithmetic

- Since pointer variables store memory addresses, we can add and subtract from them to access other addresses!
- For a pointer variable `y`, `*(y+1)` refers to the next memory location.
- Depending on the type of variable, this can be one or more bytes away.
- Caution: This must be done with extreme care!!
- If we shift the pointer to a location outside the designated memory locations then unpredictable things may happen.

Pointer Arithmetic

- Since pointer variables store memory addresses, we can add and subtract from them to access other addresses!
- For a pointer variable `y`, `*(y+1)` refers to the next memory location.
- Depending on the type of variable, this can be one or more bytes away.
- Caution: This must be done with extreme care!!
- If we shift the pointer to a location outside the designated memory locations then unpredictable things may happen.

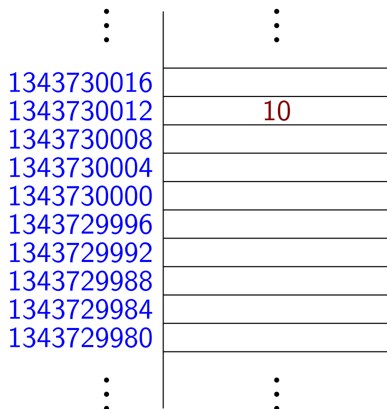
Example




```
int main()
{
    int n = 10;
    int m = 5;
    int z[2];

    for (int i=0;i<4;i++)
        z[i] = 100;
    /* Do something */
}
```

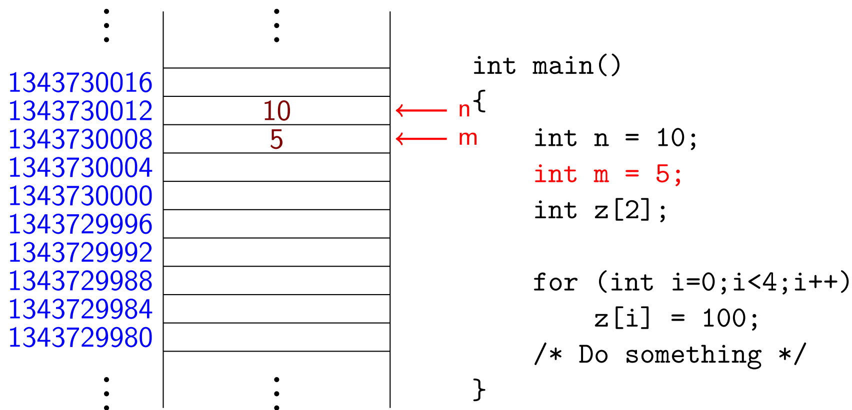
Example



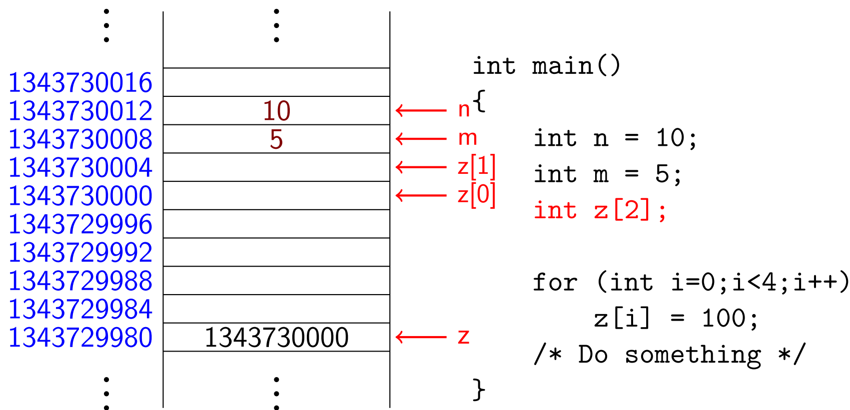

```
int main()
{
    int n = 10;
    int m = 5;
    int z[2];

    for (int i=0;i<4;i++)
        z[i] = 100;
    /* Do something */
}
```

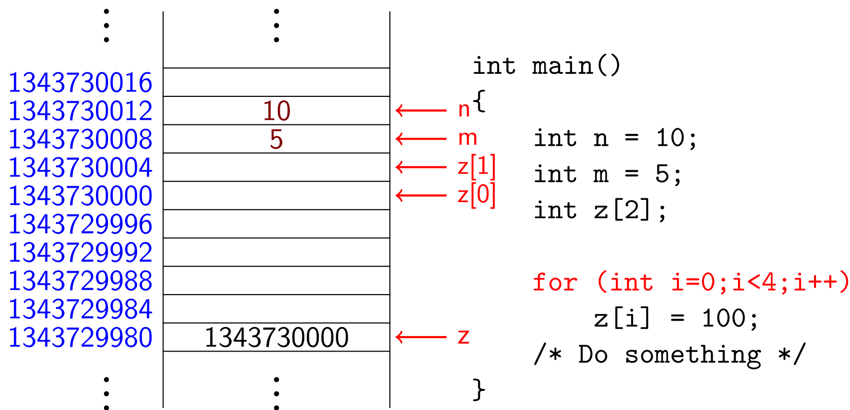
Example



Example



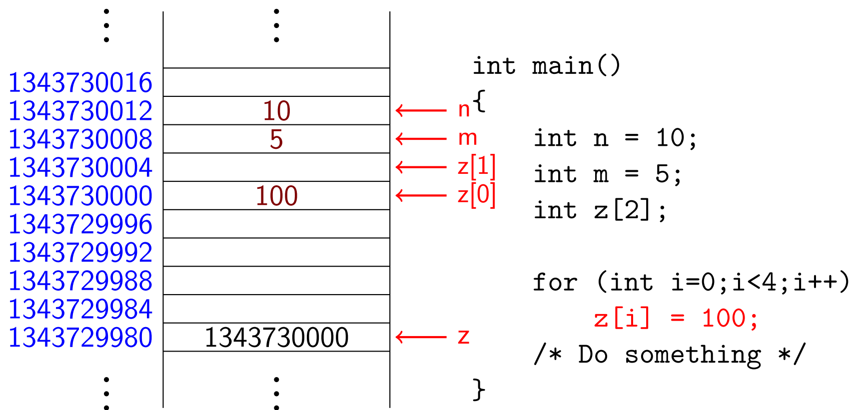
EXAMPLE



MEMORY

$i = 0$, setting z[0] or *z

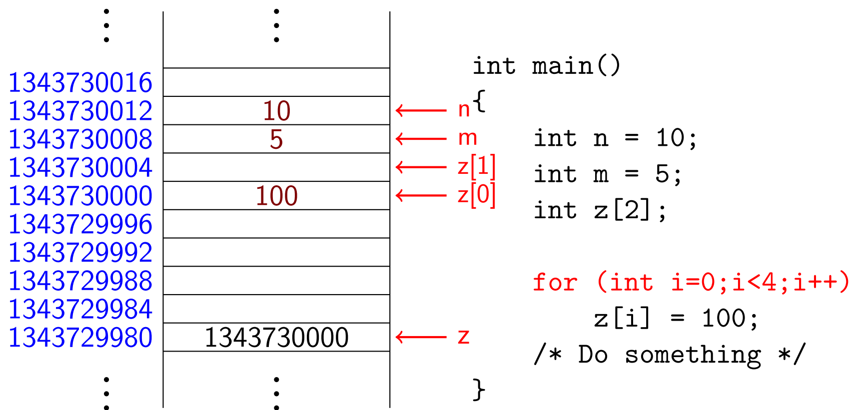
Example



MEMORY

i = 0, setting z[0] or *z

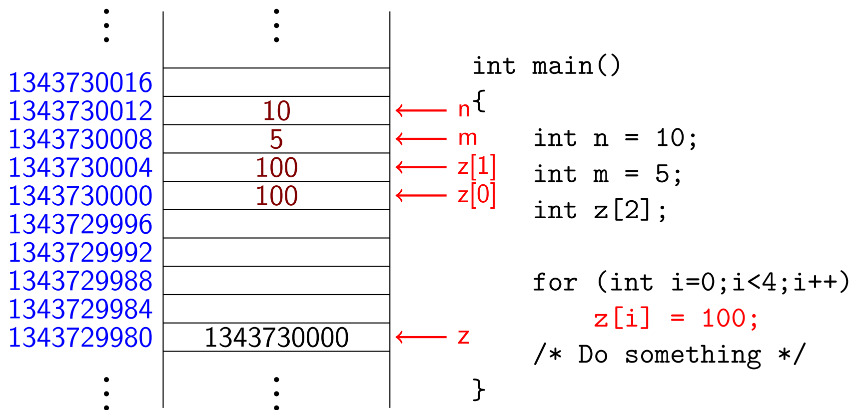
Example



MEMORY

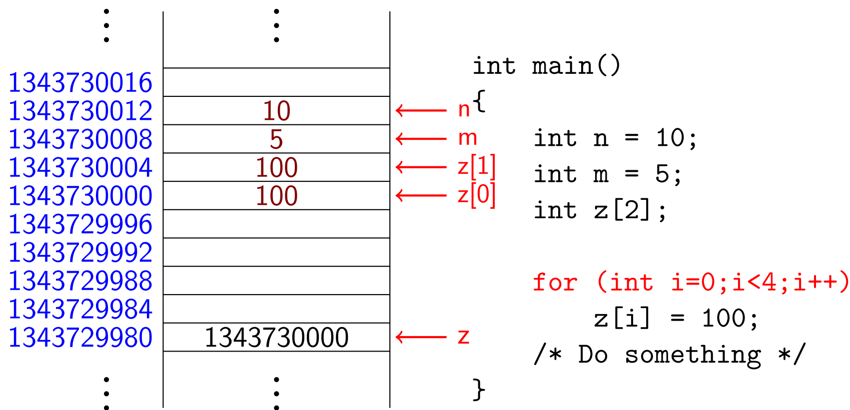
i = 1, setting z[1] or *(z+1)

Example



i = 1, setting z[1] or *(z+1)

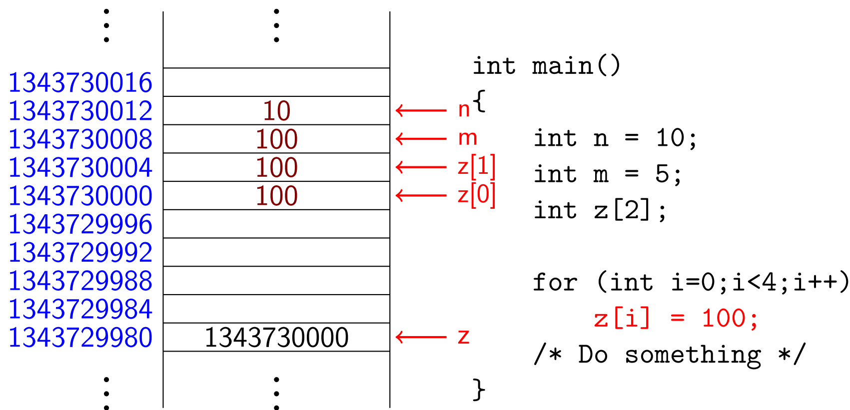
Example



MEMORY

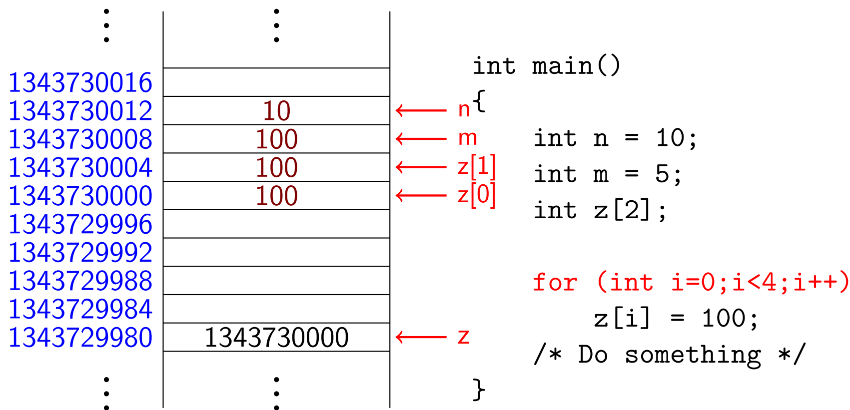
i = 2, setting z[2] or *(z+2)

Example



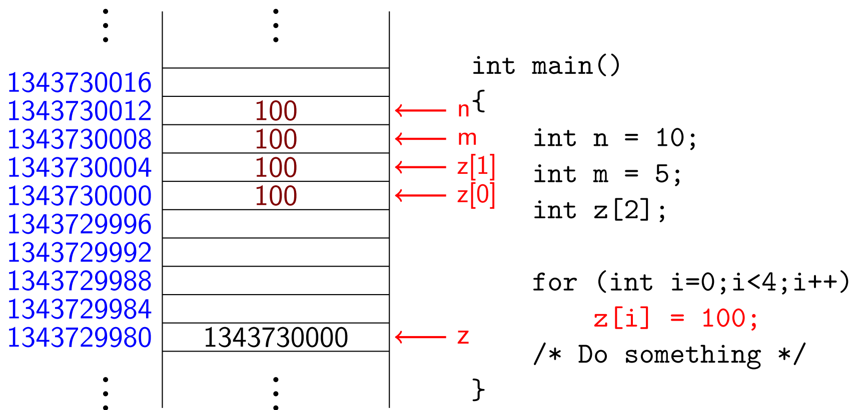
i = 2, setting z[2] or *(z+2)

Example



i = 3, setting z[3] or *(z+3)

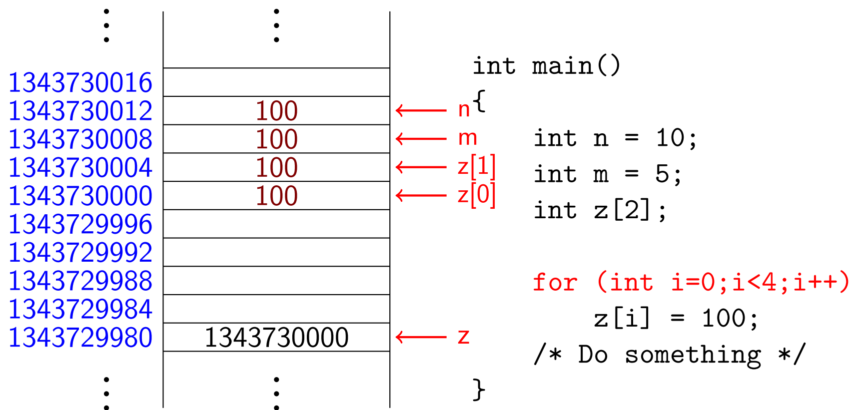
Example



MEMORY

i = 3, setting z[3] or *(z+3)

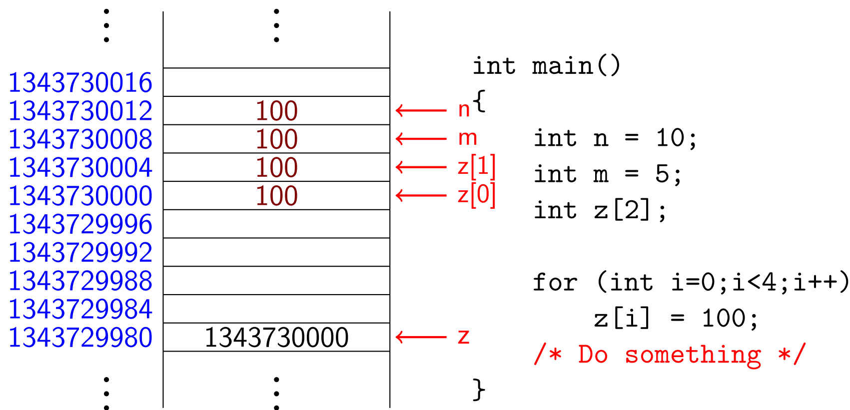
Example



i = 4

MEMORY

Example



Outline

1. Pointers
2. scanf

scanf FUNCTION

The general format of scanf is:

```
scanftf( <string constant>, argument-1, ..., argument-k )
```

scanf

- The <string constant> is a constant string specifying what needs to be read as input.
- It contains special commands, each starting with %.
- There are exactly k special commands.

scanf

- The `<string constant>` is a constant string specifying what needs to be read as input.
- It contains special commands, each starting with %.
- There are exactly k special commands.

scanf

Suppose <string constant> is:

```
"<s1>%d<s2>%c<s3>%s<s4>%f<s5>"
```

Its meaning is:

- Read string <s1>,
- Read an integer and store it in *argument-1,
- Read string <s2>,
- Read a symbol and store it in *argument-2,
- Read string <s3>,
- Read a string and store it in the array argument-3,
- Read string <s4>,
- Read a real number and store it in *argument-4, ...

scanf

Suppose `<string constant>` is:

```
"<s1>%d<s2>%c<s3>%s<s4>%f<s5>"
```

Its meaning is:

- Read string `<s1>`,
- Read an integer and store it in `*argument-1`,
- Read string `<s2>`,
- Read a symbol and store it in `*argument-2`,
- Read string `<s3>`,
- Read a string and store it in the array `argument-3`,
- Read string `<s4>`,
- Read a real number and store it in `*argument-4`, ...

scanf

Suppose `<string constant>` is:

```
"<s1>%d<s2>%c<s3>%s<s4>%f<s5>"
```

Its meaning is:

- Read string `<s1>`,
- Read an integer and store it in `*argument-1`,
- Read string `<s2>`,
- Read a symbol and store it in `*argument-2`,
- Read string `<s3>`,
- Read a string and store it in the array `argument-3`,
- Read string `<s4>`,
- Read a real number and store it in `*argument-4`, ...

scanf

Suppose `<string constant>` is:

```
"<s1>%d<s2>%c<s3>%s<s4>%f<s5>"
```

Its meaning is:

- Read string `<s1>`,
- Read an integer and store it in `*argument-1`,
- Read string `<s2>`,
- Read a symbol and store it in `*argument-2`,
- Read string `<s3>`,
- Read a string and store it in the array `argument-3`,
- Read string `<s4>`,
- Read a real number and store it in `*argument-4`, ...

scanf

Suppose `<string constant>` is:

```
"<s1>%d<s2>%c<s3>%s<s4>%f<s5>"
```

Its meaning is:

- Read string `<s1>`,
- Read an integer and store it in `*argument-1`,
- Read string `<s2>`,
- Read a symbol and store it in `*argument-2`,
- Read string `<s3>`,
- Read a string and store it in the array `argument-3`,
- Read string `<s4>`,
- Read a real number and store it in `*argument-4`, ...

scanf

Suppose <string constant> is:

```
"<s1>%d<s2>%c<s3>%s<s4>%f<s5>"
```

Its meaning is:

- Read string <s1>,
- Read an integer and store it in *argument-1,
- Read string <s2>,
- Read a symbol and store it in *argument-2,
- Read string <s3>,
- Read a string and store it in the array argument-3,
- Read string <s4>,
- Read a real number and store it in *argument-4, ...

scanf

Suppose <string constant> is:

```
"<s1>%d<s2>%c<s3>%s<s4>%f<s5>"
```

Its meaning is:

- Read string <s1>,
- Read an integer and store it in *argument-1,
- Read string <s2>,
- Read a symbol and store it in *argument-2,
- Read string <s3>,
- Read a string and store it in the array argument-3,
- Read string <s4>,
- Read a real number and store it in *argument-4, ...

scanf

Suppose `<string constant>` is:

```
"<s1>%d<s2>%c<s3>%s<s4>%f<s5>"
```

Its meaning is:

- Read string `<s1>`,
- Read an integer and store it in `*argument-1`,
- Read string `<s2>`,
- Read a symbol and store it in `*argument-2`,
- Read string `<s3>`,
- Read a string and store it in the array `argument-3`,
- Read string `<s4>`,
- Read a real number and store it in `*argument-4`, ...

`scanf`

Suppose `<string constant>` is:

```
"<s1>%d<s2>%c<s3>%s<s4>%f<s5>"
```

Its meaning is:

- Read string `<s1>`,
- Read an integer and store it in `*argument-1`,
- Read string `<s2>`,
- Read a symbol and store it in `*argument-2`,
- Read string `<s3>`,
- Read a string and store it in the array `argument-3`,
- Read string `<s4>`,
- Read a real number and store it in `*argument-4`, ...

scanf EXAMPLES

For the call:

```
scanf("Number = %d, string = %s", &n, str);
```

the input must be one of the following:

```
Number = 35, string = test
Number       =      35, string         =     test
Number=35,string=test
```

The following inputs are wrong:

```
number = 35, string = test
Number = 35, string test
```

scanf EXAMPLES

For the call:

```
scanf("Number = %d, string = %s\n", &n, str);
```

the input must be of the following form:

```
Number = 35, string = test         <some non-whitespace>
```

This is because the `\n` at the end of format string matches with any number of whitespaces.

scanf EXAMPLES

Consider the program:

```
main()
{
    char str[10];

    scanf("%s", str);
    printf("%s", str);
}
```

- Suppose its input is: abcdefghijklmnop, which is a 16 character long string.
- This may well be accepted by scanf and stored in the array str!
- The reason is the same as before: if the extra memory locations are inside the program, then those locations will be overwritten.

scanf EXAMPLES

Consider the program:

```
main()
{
    char str[10];

    scanf("%s", str);
    printf("%s", str);
}
```

- Suppose its input is: `abcdefghijklmnop`, which is a 16 character long string.
- This may well be accepted by `scanf` and stored in the array `str`!
- The reason is the same as before: if the extra memory locations are inside the program, then those locations will be overwritten.

scanf EXAMPLES

Consider the program:

```
main()
{
    char str[10];

    scanf("%s", str);
    printf("%s", str);
}
```

- Suppose its input is: `abcdefghijklmnop`, which is a 16 character long string.
- This may well be accepted by `scanf` and stored in the array `str`!
- The reason is the same as before: if the extra memory locations are inside the program, then those locations will be overwritten.

scanf Examples

Consider the program:

```
main()
{
    char str[10];

    scanf("%s", str);
    printf("%s", str);
}
```

- Suppose its input is: `abcdefghijklmnop`, which is a 16 character long string.
- This may well be accepted by `scanf` and stored in the array `str`!
- The reason is the same as before: if the extra memory locations are inside the program, then those locations will be overwritten.