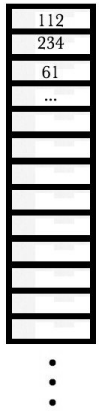
The C Programming Language

First, “machine language”...

Memory is a vast array of bytes:



Addresses:

We call this “main memory”
(as opposed to “secondary storage”)

```
LOAD 1240, R2
ADD 2000, R2
STORE R2, 3000
```

```
z = x + y
```

- compilation
- interpretation

```
i = 0;
while (i < 10) {
    stuff goes here
    i++;
}
```

nasty because the loop control information is scattered

```
for (i = 0; i < 10; i++) {
    stuff goes here
}
```

`++` and `--`

`i++` and `++i`

```
j = i++;
j = ++i;
j = ++i + 28;
```

If the variable `i` previously had the value 4, then "`++i`" increments `i` (so that it's now 5), and the value of that expression "`++i`" is 5.

If the variable `i` previously had the value 4, then "`i++`" still increments `i` (so that `i` again becomes 5); however, the value of that expression "`i++`" is 4.

```
#include <stdio.h>

int main()
{
    int i;
    extern int gcd(int x, int y);

    for (i = 0; i < 20; i++)
        printf("gcd of 12 and %d is %d\n", i, gcd(12, i));
    return(0);
}

int gcd(int x, int y)
{
    int t;

    while (y) {
        t = x;
        x = y;
        y = t % y;
    }
    return(x);
}
```

```
#include <stdio.h>

int main()
{
    int i;
    extern int gcd(int x, int y);

    for (i = 0; i < 20
        printf("gcd of
    return(0);
}

int gcd(int x, int y)
{
    int t;

    while (y) {
        t = x;
        x = y;
        y = t % y;
    }
    return(x);
}
```

"declaration": tells the compiler how to access the item

"definition": creates the item (and in C, is also a declaration, but not necessarily vice versa)

```
#include <stdio.h>

int main()
{
    int i;
    extern int gcd(int x, int y);

    for (i = 0; i < 20; i++)
        printf("gcd of 12 and %d is %d\n", i, gcd(12, i));
    return(0);
}

int gcd(int x, int y)
{
    int t;

    while (y) {
        t = x;
        x = y;
        y = t % y;
    }
    return(x);
}
```

- the function body is delimited by braces
- braces enclose a "block"
- the bodies of most control constructs are defined as "statement or block"
- braces introduce a scope level

Declaration with initialization:

```
int i = 0;
```

(only if you're going to use that value!)

An uninitialized local variable has no default value, and to use its value before assigning one is an error. Furthermore, the error is not diagnosed!

In general, no run-time checking.

```
#include <stdio.h>

int main()
{
    int i;
    extern int gcd(int x, int y);

    for (i = 0; i < 20; i++)
        printf("gcd of 12 and %d is %d\n", i, gcd(12, i));
    return(0);
}

int gcd(int x, int y)
{
    int t;

    while (y) {
        t = x;
        x = y;
        y = t % y;
    }
    return(x);
}
```

C uses "int" as the boolean type.
Boolean operators generate 1 for true or 0 for false.
Any non-zero value counts as "true" in a condition.

```
#include <stdio.h>

int main()
{
    int i;
    extern int gcd(int x, int y);

    for (i = 0; i < 20; i++)
        printf("gcd of 12 and %d is %d\n", i, gcd(12, i));
    return(0);
}

int gcd(int x, int y)
{
    int t;

    while (y) {
        t = x;
        x = y;
        y = t % y;
    }
    return(x);
}
```

- no polymorphism
- no objects