hw.java

or

or


Installing gcc

- Windows: install Cygwin with **gcc-core** and **libmpfr1** packages
- Mac OS X: install Apple developer tools
- Linux: install **gcc** package
C Program that Succeeds at Nothing

```c
int main() {
    return 0;
}
```
Compile and Run

```
% gcc x.c
% ./a.out
```

under Windows, it’s a.exe instead of a.out

```
% gcc -o x x.c
% ./x
```

under Windows, it’s actually x.exe, but just x works
C Program that Fails at Nothing

```c
int main() {
    return 1;
}
```

a non-0 result reports failure
Enabling Warnings

```
% gcc -Wall -o x x.c
% ./x
```
C Program that Prints, But Makes gcc Complain

```c
int main() {
    printf("Hi\n");
    return 0;
}
```

*Language within a language:* Inside a string, \n means “newline” — and that’s true for C, Java, Racket, and most languages.
C Program that Prints, And Keeps gcc Happy

```c
#include <stdio.h>

int main() {
    printf("Hi\n");
    return 0;
}
```

**Include is similar to Require or Import**
C Program that Prints a Number

```c
#include <stdio.h>

int main() {
    printf("Ten and ten make %d\n", 10+10);
    return 0;
}
```

*Language within a language within a language:* In a string passed to `printf`,

- `%d` means “print the next integer”
- `%f` means “print the next double”
- `%s` means “print the next string”
- `%p` means “print the next address”
- `%c` means “print the next character”
Hexadecimal Numbers

```
#include <stdio.h>

int main() {
    printf("Hex 10 and hex 10 make %d\n", 0x10 + 0x10);
    return 0;
}
```

0x starts a base-16 number
Everything is a Number

#include <stdio.h>

int main()
{
    printf("%p %p\n", main, printf);
    return 0;
}
Variables Live in Memory

#include <stdio.h>

int main() {
    int v = 5;

    printf("At %p is %d\n", &v, v);
    return 0;
}

& as an operator means “the address of”
&v

0x10756

V at 0x10756

5
#include <stdio.h>

int main()
{
    int v = 5;
    int* p = &v;

    v = 6;
    printf("At %p is %d\n", p, *p);
    return 0;
}

* in a type means “the address of a”

* as an operator means “value at the address”
Changing Memory can Change Variables

#include <stdio.h>

int main() {
    int v = 5;
    int* p = &v;

    *p = 7;
    printf("V at end: %d\n", v);
    return 0;
}
Array Notation Also Looks in an Address

```c
#include <stdio.h>

int main() {
    int v = 5;
    int* p = &v;

    printf("At %p is %d\n", p, p[0]);
    return 0;
}
```
```
#include <stdio.h>

int main() {
    int v = 5;
    int* p = &v;

    printf("At \%p is \%d\n", p+1, p[1]);
    return 0;
}

The result is unpredictable
```
#include <stdio.h>

int main() {
    int a[3] = { 1, 2, 3 };  
    int* p = a;  

    printf("%d, %d, %d\n",  
            a[0], p[1], *(p + 2));  
    return 0;  
}  

Copy
Array Names Are a Little Strange

#include <stdio.h>

int main() {
    int a[3] = { 1, 2, 3 };
    int* p = a;
    int* q = &a;

    printf("%p = %p, but not %p\n", p, q, &p);
    return 0;
}

Special treatment of sized-array names makes []-expression notation consistent
A String is an Array of Characters

```c
#include <stdio.h>

int main() {
    char* s = "apple";

    printf("%s: %c, %c, %c\n",
           s, s[0], s[1], *(s + 3));
    return 0;
}
```
Characters are Just Numbers

```c
#include <stdio.h>

int main() {
    char* s = "apple";

    printf("%s: %d, %d, %d\n",
           s, s[0], s[1], *(s + 3));
    return 0;
}
```
#include <stdio.h>

int main() {
    char* s[3] = { "apple",
                  "banana",
                  "coconut"  };

    char **ss = s;

    printf("%s (%c...), %s, %s\n",
            ss[0], ss[0][0], ss[1], ss[2]);
    return 0;
}
Using Command-Line Arguments

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

int main(int argc, char** argv) {
    int a, b;

    a = atoi(argv[1]);
    b = atoi(argv[2]);

    printf("%d\n", a + b);

    return 0;
}
```
Sizes of Numbers

Each "box" in your machine’s memory holds a number between -128 and 127

or 0 to 255, depending on how you look at it

• a **char** takes up one of them

• a **short** takes up two of them (-32768 to 32767)

• an **int** takes up four of them (-2147483648 to 2147483647)

• a **long** takes up four or eight, depending

• an address takes up four or eight, depending
  **char***, **int***, **char****, etc.
#include <stdio.h>

int main() {
    char cs[2] = {0, 1};
    int is[2] = {0, 1};

    printf("Goes up by 1: %p, %p\n", cs, cs+1);
    printf("Goes up by 4: %p, %p\n", is, is+1);

    return 0;
}

Copy
Computing Sizes

#include <stdio.h>

int main()
{
    char cs[2] = {0, 1};

    printf("char size is %d\n", sizeof(char));
    printf("char size is %d\n", sizeof(cs[0]));
    printf("cs size is %d\n", sizeof(cs));
    printf("address size is %d\n", sizeof(&cs));
    return 0;
}

The sizeof operator works on types or variables
Allocation

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

int main() {
    int* a;

    a = (int*)malloc(100 * sizeof(int));
a[99] = 5;
    printf("array at %p ends in %d\n", a, a[99]);

    return 0;
}
More C: For Loops

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

int main(int argc, char** argv) {
    int i;
    int sum = 0;

    for (i = 1; i < argc; i++) {
        sum += atoi(argv[i]);
    }
    printf("%d\n", sum);

    return 0;
}
```

... just like Java
More C: Defining Functions

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

int twice(int n) {
    return n + n;
}

int main(int argc, char** argv) {
    printf("%d\n", twice(atoi(argv[1])));
    return 0;
}
```

... just like Java