CS 354 : The Other Room

Last Time:

Computer works:

CPU:

- Fetch instruction
- Decode (figure out what inst does)
- Execute
- ~~Get r~~

Bits: 1's, 0's

byte: 8 bits

Memory: simple

byte-addressable

Towards C

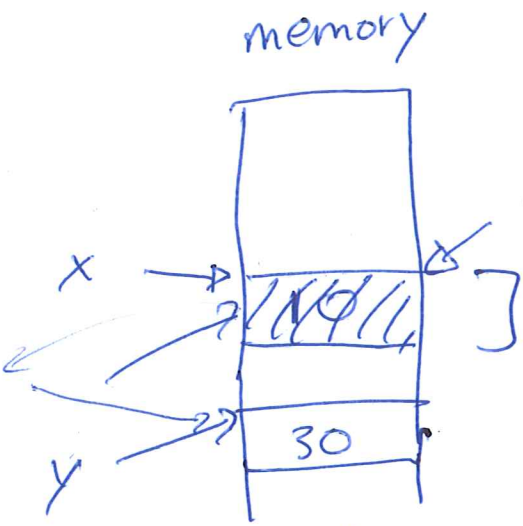
=> whole Numbers (Integers)

32-bits (4 bytes) C int (not a math integer)

=> Efficiency + Control

Variables

```
int x;
```



operators:

* / %

+ +

Assignment: = operator

```
x = expression;

x = 10;

int y;
y = x + 20;
```

```
x = 10;   — S1
y = 20;   — S2
z = x + y; — S3
```

C Instruction Pointer

C Abstract Machine

Comparisons

Boolean values: True, False

C represents: 1 (non-zero), 0 zero

$(3 > 2) \Rightarrow 1$

&& (ampersand) $\Rightarrow$ AND

$(a > 10) \;\&\&\; (a < 20)$

(Boolean Logic)

New operators: ~~to~~ logical operators

binary [AND, OR]

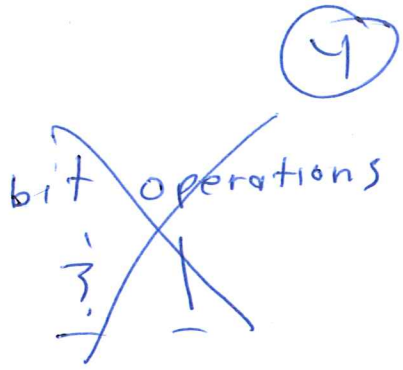
unary [NOT]

Truth Tables

X	NOT
F	T
T	F

X	Y	AND	OR
F	F	F	F
F	T	F	T
T	F	F	T
T	T	T	T

AND &&
OR ||
NOT !



Precedence : google C precedence

Make Decisions : if expr is true, exec. compound stmt

```
if (expression) {
    statement#1;
    stmt 2;
    :
}
```

compound statement

```
if (expression)
    statement1;
```

if (expr) {

≡

} else {

≡

}

other:
switch stmt
(skip for now)

short circuit

if (x && y) {

}

short circuit:
if x is false
⇒ (x && y) ⇒ false
don't evaluate y

Decisions : if

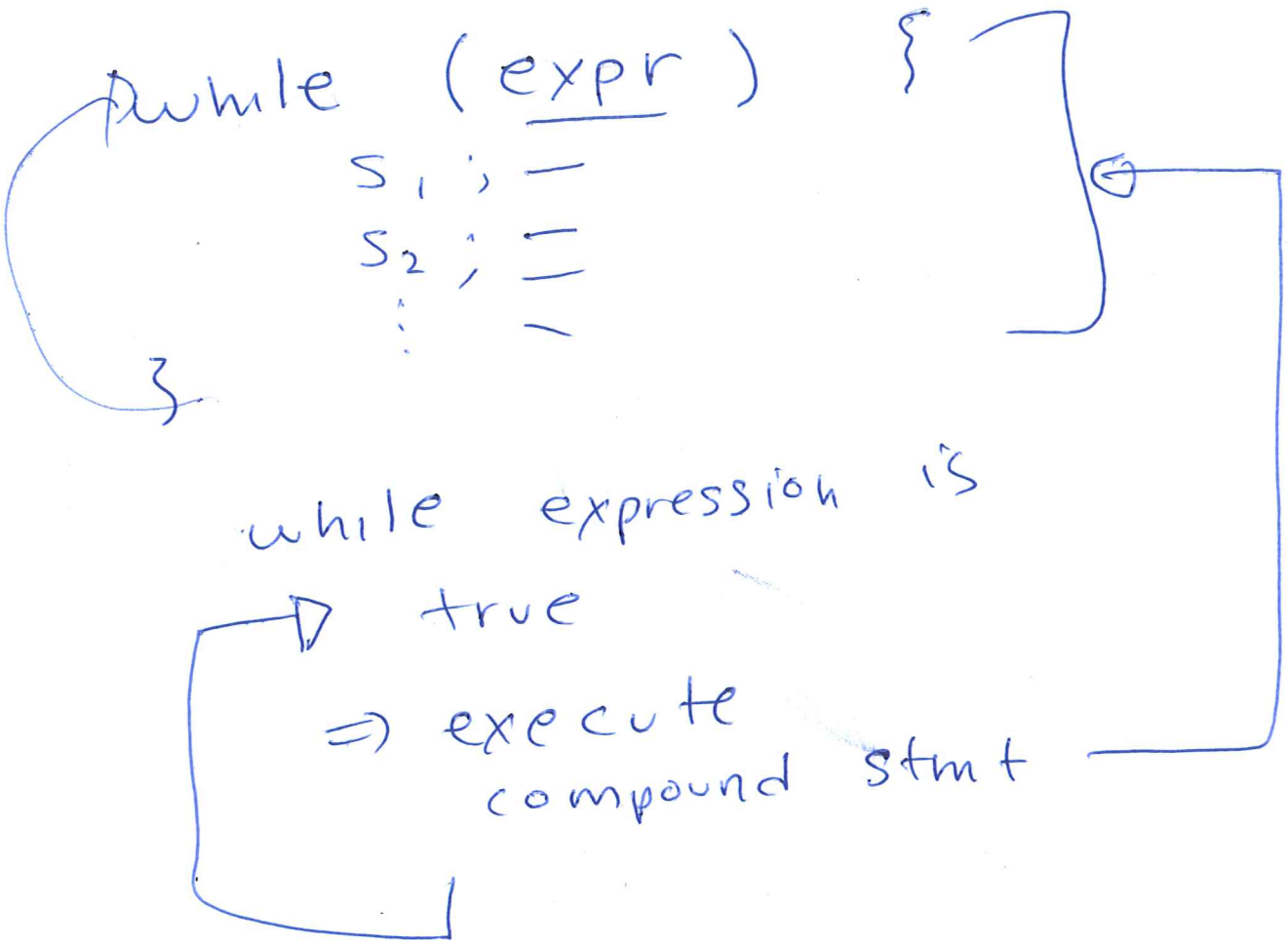
Repetition : Looping

while for

```
while (expr) {
    S1;
    S2;
    :
}
```

while expression is true

⇒ execute compound stmt



for (initial expression ; expr ; post-block stmt) {

⇓

for (init stmt ; expr ; post-stmt) {

=

}

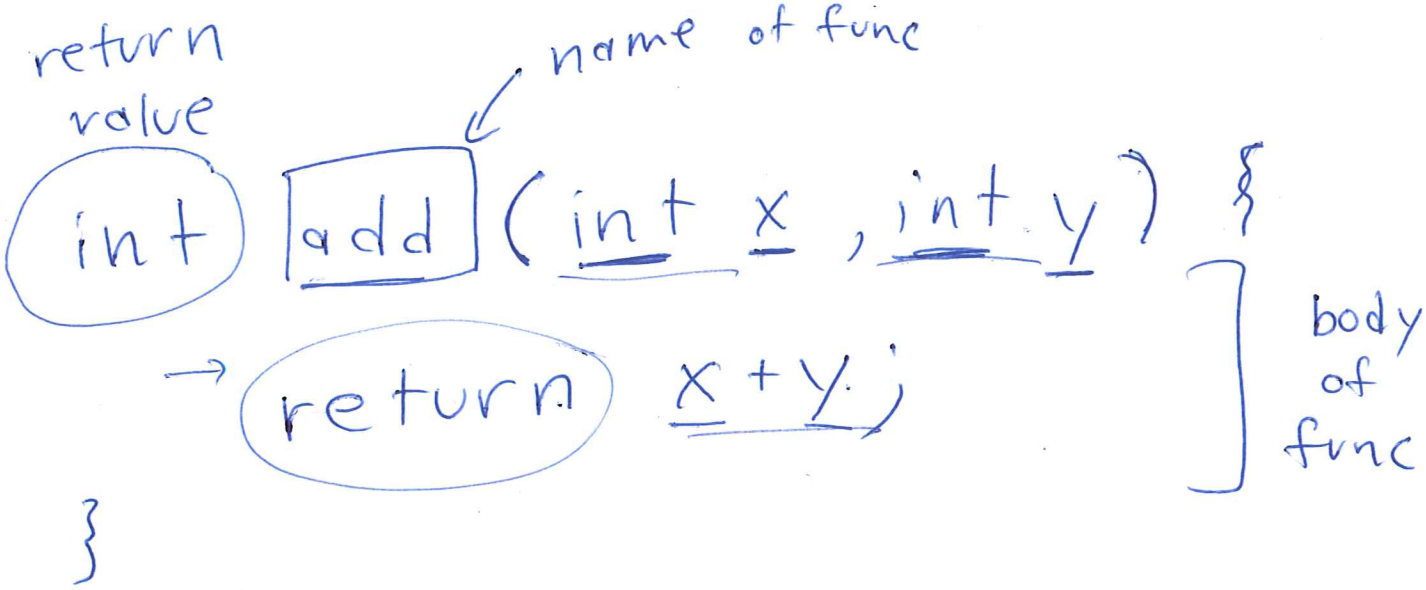
7

⇓

```
init stmt;
while (expr) {
    =
    =
    post-stmt;
}
```

Functions

~ objects ~~data + methods~~




```
int foo() {
    return 0;
}
```

calling a function:

```
int a = 10;
int b = 11;
int c = add(a,b);
```

function call operator

(c == 21) => true

How to call a function and actually change variable value in caller

→ Pointers (Addresses)

→ The Stack

→ Scope of variable

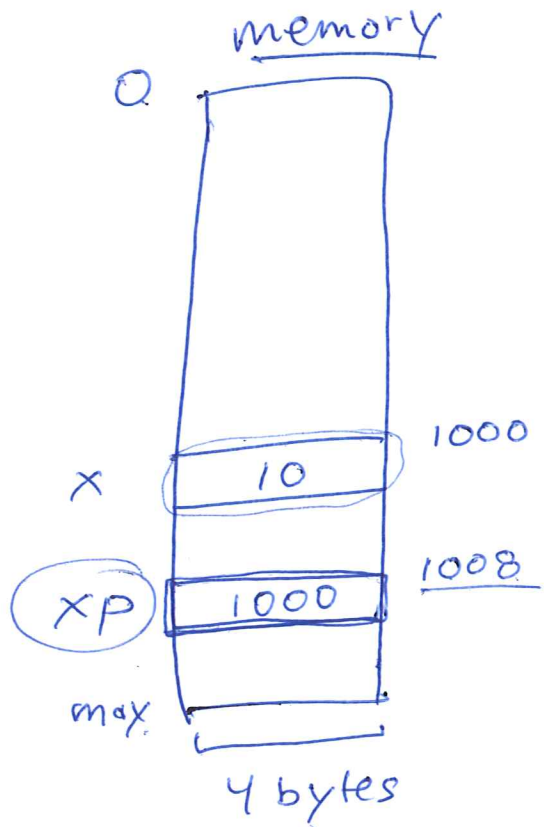
Pointers:

```
int x;
x = 10;
int* xp;
xp = &x;
```

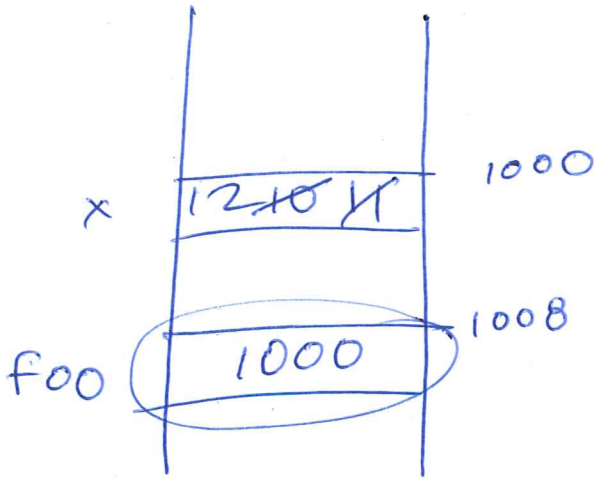
int* xp; — pointer to integer

&x — address of x

pointer: address



```
int x;
x = 10;
int* foo;

foo = &x;

*foo = 11;

*foo = 12;
```



Pointers (Stack) Scope

```
int x = 10;
```

```
int* xp;
```

pointer to int

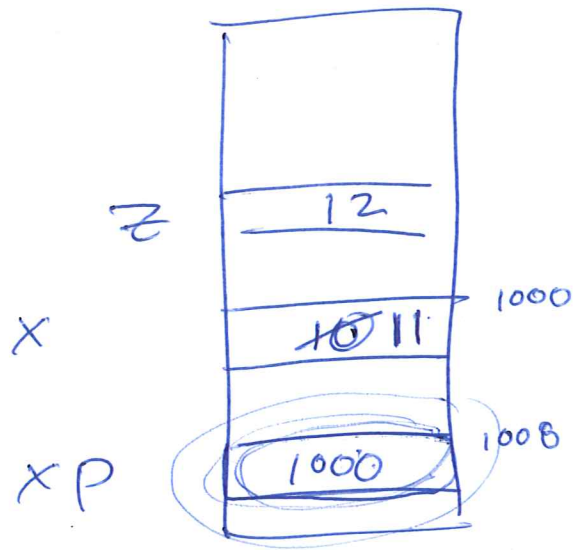
```
int *xp;
```

```
xp = &x;
```

new operator:

=> *

dereference




```
*xp = 11;
```

x = 10;

```
int z = *xp + 1;
```

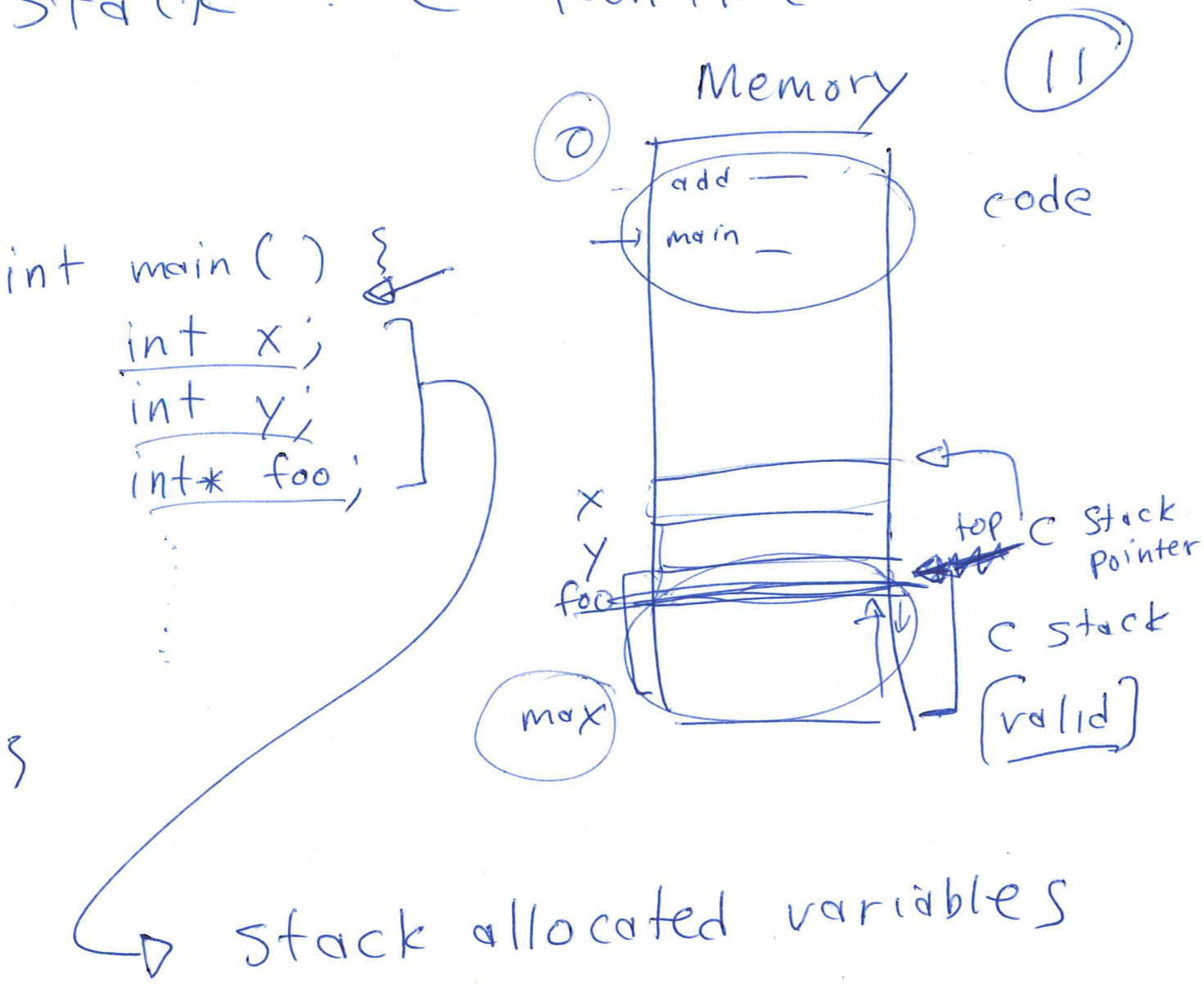
reads

int

```
= 2 *xp + 1;
= 2 * *xp + 1;
```

```
y = 2x;
y = 2*x;
```

Stack : C Runtime Stack



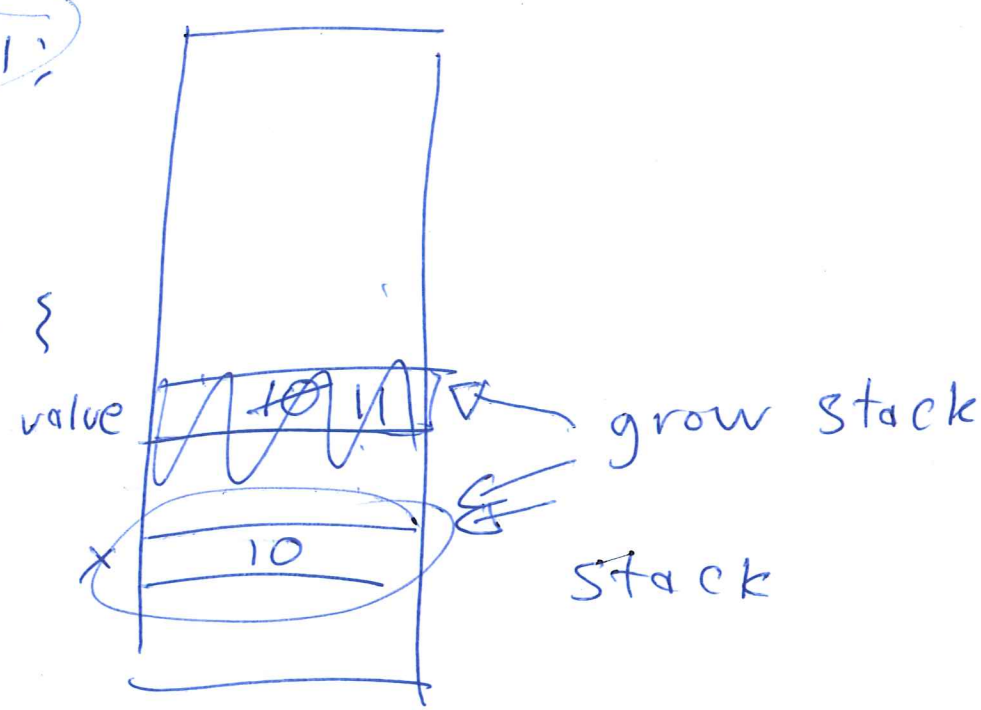
stack allocated variables

allocated when function is entered

deallocated when function returns

"automatic" variables

```
void
inc (int value) {
    value = value + 1;
}

int main () {
    int x = 10;
    inc (x);
}
```



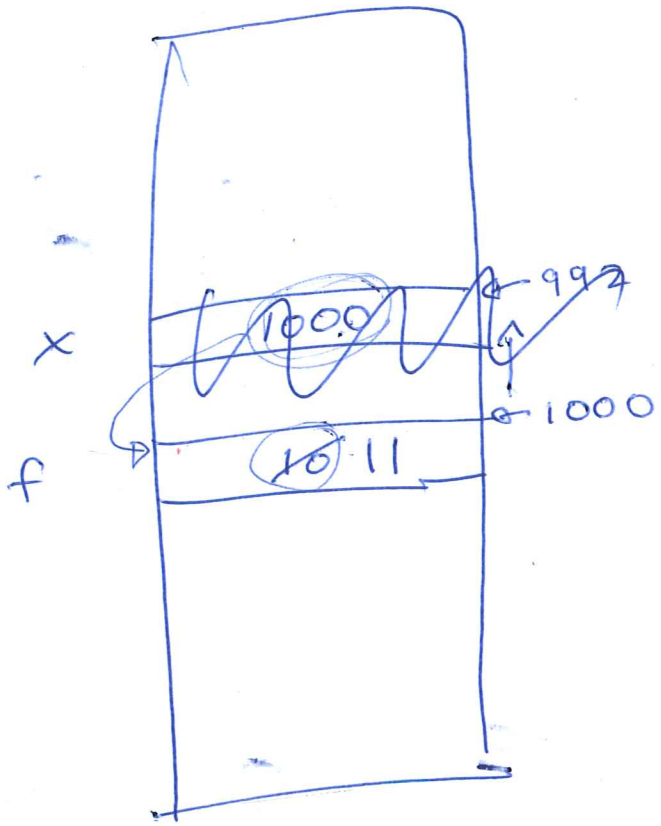
variable

Scope

=> when name valid?

```
void inc (int* x)
{
    *x = *x + 1
}
```

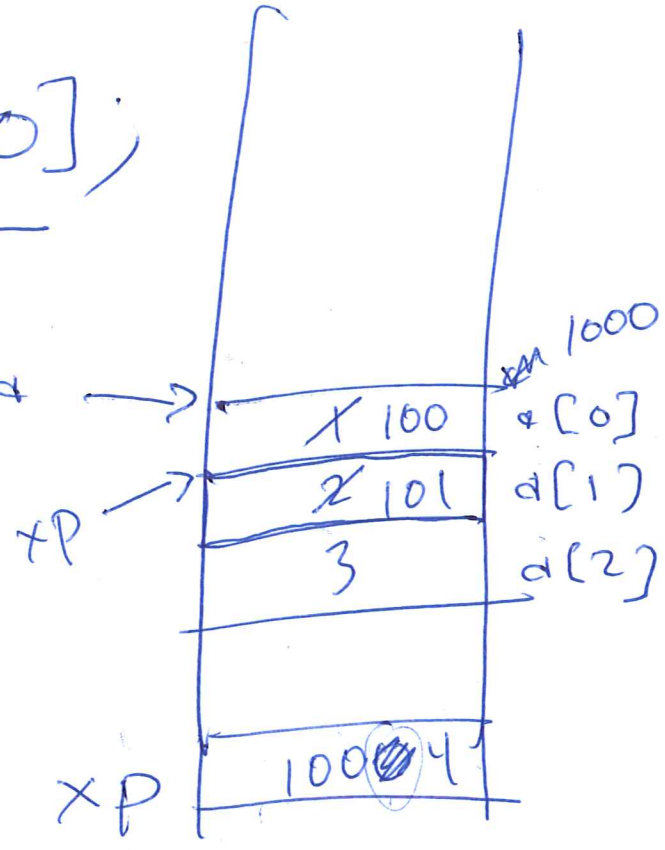
```
main ()
{
    int f = 10;
    inc (&f);
    show (f); => 11
```

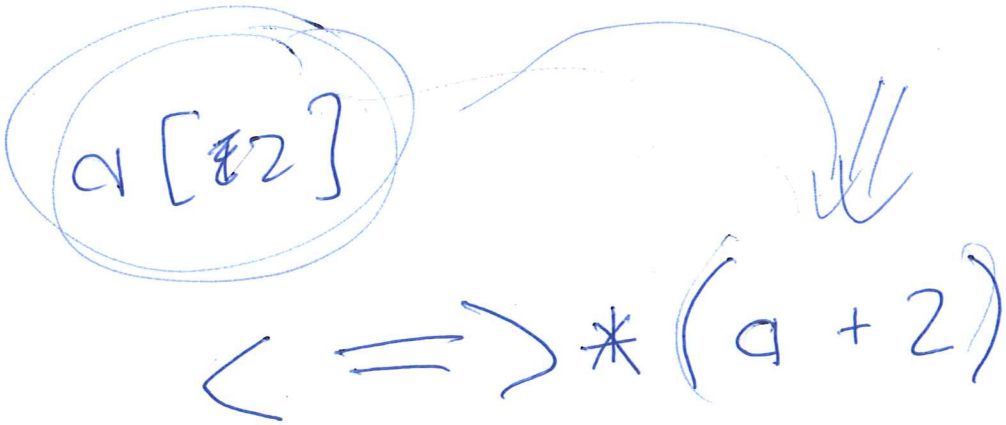


Array / Pointer "equivalence"

```
int a[3];

int* xp;

xp = a[0];

*xp = 100;

xp = xp + 1;

*xp = 101;
```

pointer arith




a[#2]
<==>*(a+2)

More Types :

character

```
char x;
```

1 byte

numeric value

0 255

ASCII

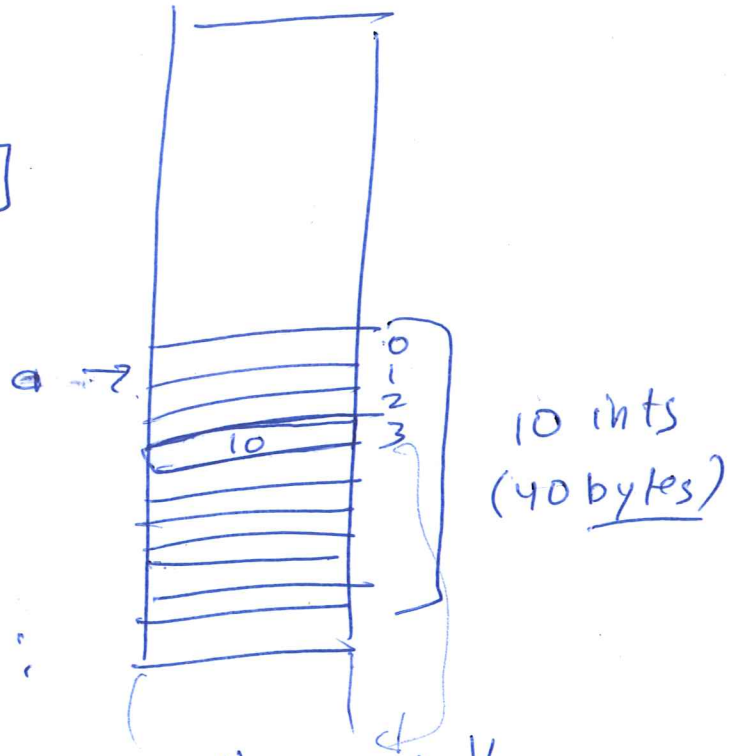
Array

array of integers

```
int a[10];
```

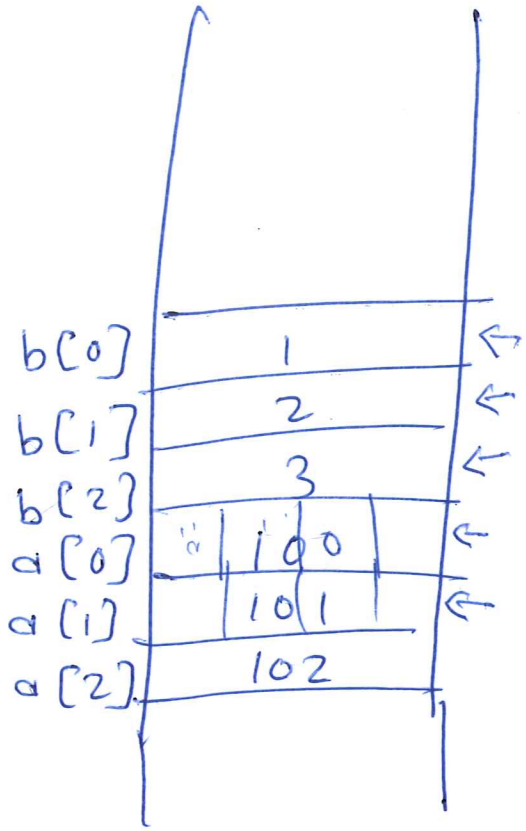
new operator: []

```
a[3] = 10;
```



weak type in C:

a[23] ⇐ will "work"


b[0]
1
b[1]
2
b[2]
3
a[0]
100
a[1]
101
a[2]
102

string: new type

char array + a little bit