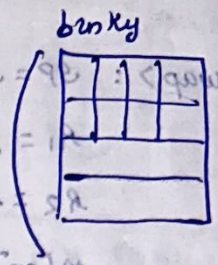
→ why public has access to winky, slinky → ∵ it has address! object!

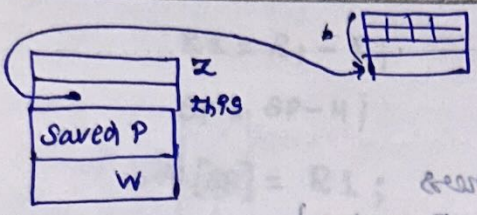
→

```
int n=17;
binky b;
b.minky(&n);  ⟶  // binky::minky(&b, &n)
```

Actually: long (behind the scenes).



Just creates object b, [minky(Source), &b - branch]



c++] Assembly level: same!
c

Static → method →

```
class fraction {
    public
        fraction (int x, int d = 1);
    {
        void reduce ();
    }
    public
        static int gcd (int x, int y);
```

→ Don't need an instance to invoke it (standalone)

* Linking: all files (.o) → stack them one another → bundle.

```
#define kWidth 480
#define kHeight 720
{
printf("width: %d", kWidth);
printf("Area = %d", kWidth * kHeight);
```

Preprocessor: Read .c

(define) → read words (include to hash sets).

Just put (replace) token!

doesn't even know as number!

Say: #define kWidth 480 rst

Takes: But: error appears during compilation (used for multiply)

"error" → Can't do that!

Lecture-12

```
#define kwidth 40
#define klength 80
#define kperimeter 2* (kwidth * klength)
#define MAX(a,b) ((a)>(b)?(a):(b))
```

macros → use (Son good).

a and b can be int, float, ...

```
MAX (40.2, "Hello");
```

→ compilation (error!)

MAX (10,40) → 10 > 40 : b
replaced by 40

Text & replace — preprocessing

* int max = MAX (fib (in), fact (4000)); → problem: (code style: (continuation)

```
#define NthElemAddr (base, elemSize, index)
    ((char *)base + index * elemSize)
```

→ without assert!

→ no type checking!

```
void * vectorNth (vector *v, int position)
{
    assert (position >= 0);
    assert (position < v->logLength);
    return NthElemAdd (v->elems, v->elemSize, position);
}
```

But this do!

assert → also a macro!

upcasting: (char *) → (void *)

more specifically typed pointer (to) generic.

* downcasting! (void *) → Some specific!

```
#define assert (cond)
    (cond) ? ((void) 0) : fprintf (stderr, "------");
                                exit();
```

cast 0 to be a void [don't assign]!

File name, etc.

```
#ifdef NDEBUG
    #define assert (cond) (void) 0
#else
    #define assert (cond) \
        (cond) ?((void) 0) : fprintf (stderr, "____"); exit(0)
#endif
```

→ ((void)0) → harmless statement! [0 lines of assembly]

int max = MAX (fib(n), fact(4000)); why?

↳ call it twice!

int max = (fib(n)) > (fact(4000)) ? fib(n) : fact(4000);

"calculated twice" – very time consuming!

int larger = MAX (m++, n++);

(m++) > (n++) ? (m++) : (n++)

2 times done → we need only once!

* Finding where is the problem: in 10 million lines – not so easy!

"do it the right way, first time"

#include <assert.h> → .h file → system header file

#include "vector.h" → may be client written (search other files).

"write all prototypes in a .h file"

∴ header has header which has header → possible!

→ gcc -E vector.c [stop at preprocessing]

#ifndef -vector.h- (define)

#define -vector.h-

:

#endif

Sometimes multiple time vector.h may be define so #ifndef

"so reduced work"

"we can't declare storage for anything → in .h file

occasionally : global variables?

↓

no need: global variables are available!

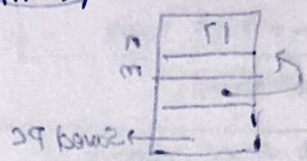
.h → Just no code generation - Just definition!

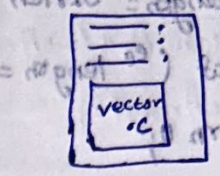
→ no value: Including twice - Just waste of time.

→ *pragma

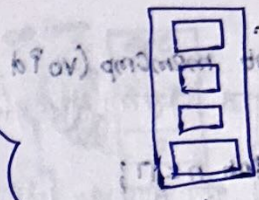
```
#include <stdio.h>   //printf
#include <stdlib.h>  //malloc, free
#include <assert.h>

int main (int argc, char *argv[])
{
    void *memory = malloc(40);
    assert (memory != NULL);
    printf ("Got it\n");
    free (memory);
    return 0;
}
```





.h .h .h .c



↓ Compile

↓ asc

↓ .o (vector.o)

.o file

⋮

CALL <malloc>

CALL <printf>

CALL <free>

RV = 0

RET

a.out (unless -o flag used)

Say -o myprog

myprog (Link phase - executable)

→ assert can't be in .o

∴ macro → replaced by something right after identification.

Default (IPIRK) Some libraries: printf, fprintf, malloc, calloc, realloc, free.

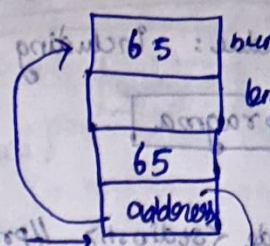
If I forget: Stdio.h → Then what is printf? → Can't build!

Lecture - 13

→ may warn: but still runs: because: link always links std. libraries

→ forget <assert.h> → assert (macro) - only defined in assert.h (not in any std. library).

```
int main()
{
  int num=65;
  int length= strlen ((char *) & num , num);
  printf ("length = %d \n", length);
  return 0;
}
```



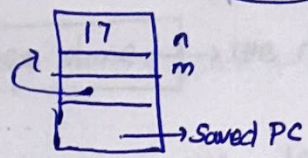
(char *)

→ int strlen (char *s , int len);

→ int memcmp (void *v1); ⟶ Changing prototype creates mess!

int memcmp (void *v1, void *v2, int size);

```
{
  int n=17;
  int m=memcmp (&n);
}
```



★ abc → compiled with the idea → 1 arg!

★ when executing - collapsation starts! ⟶ Crash!

→ strongly typed → need to be right (compilation)

→ C++ (compile) → Actually doesn't tag great info as the name uses name of func, datatypes (to assemble a complicated function symbol

CALL <memcmp> ⟶ C

CALL <memcmp - void - P> ⟶ like this! [Safer]

⟶ seg fault → dereference a bad pointer! ★(NULL)

⟶ NULL → to call var, → malloc] No! why dereferencing!

→ Bus error: dereference an address - really in somewhere in one of the four segments → address not correspond to what you're thinking!

void *vp; ∴ vp → unassigned

*(short *) vp = 7; (Null pointer)

All int → begin with address → multiple of 4.

begin (short) → even address

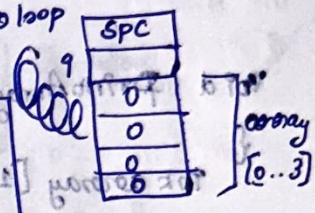
OS restriction! (Clean, optimize)

can't dereference → odd address.

*(int *) VP = 55; (Bus error)! ∴ wasn't multiple of 4.

```
int main()
{ int i;
  int array[4];
  for (i=0; i<=4; i++)
  {  array[i] = 0;
  }
}
```

→ overflow: bounds of array!



array[4] → editing → makes it 0

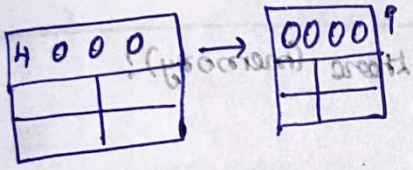
Now i = 0, → ∞ loop

Complicated version of while(1).

mycompiler: No error (may be not saved this way!)

Theory: runs forever!

x86 machines - little endian

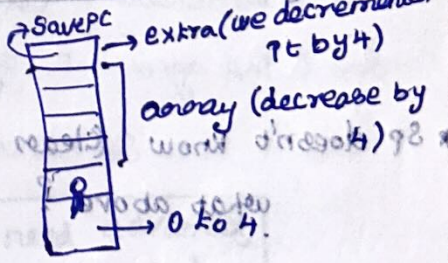


Big indian - may run good!

∞ loop

```
void foo()
{
  int array[4];
  int i;
  for (i=0; i<=4; i++)
  {  array[i] = 4;
  }
}
```

∴ uninitialized array slot!



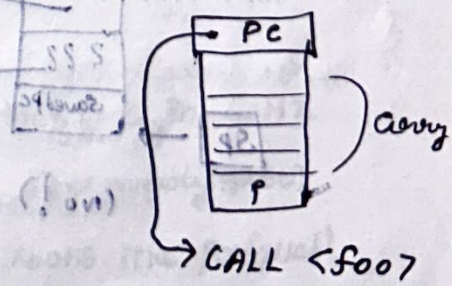
∴ decrement by 4 → foo-4

∞ recursion ∴ Something else! (Impact! : maybe error!)

"Buffer overflow"

calls foo again!

Lecture-14



"decrement Save PC" - not good -

make it PC-4

goto previous line!

Again & again run - recursively!

Some Architecture: don't put save PC — right after (above)!

```
int main(—, —)
{
    DeclareAndInitArray();
    PrintArray();
}
```

```
void DeclareAndInitArray();
{
    int array[100];
    int i;
    for (i=0; i<100; i++){
        array[i] = i;
    }
}
```

→ Not returning (to anywhere) when it returns — doesn't clean! will be still there! (memory)!

```
void PrintArray()
{
    int array[100];
    int i;
    for (i=0; i<100; i++)
        printf("%d", array[i]);
}
```

→ printing 0 to 99 (correct)

why: when function gets done still there (memory)!

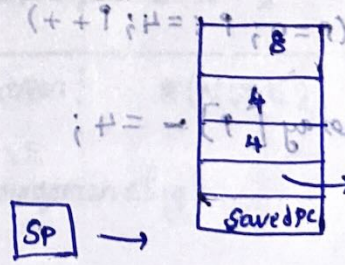
→ channeling: → how memory is laid out

```
int printf(const char * control, ...);
```

[Gcc — typecheck — for given & %d (mentioned)]

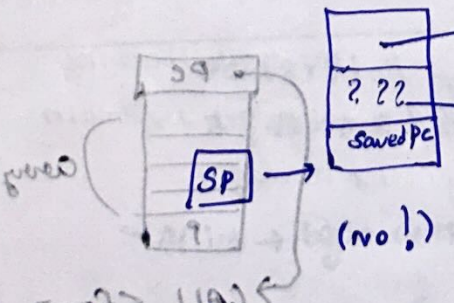
★ why stack (decrementing: setup!)

★ SP (doesn't know (clean info) about what above?)



★ control string — read char by character

★ If no control string → 8, 4, 4 are taken as char★



→ no constant — to get to control string (may be any length)

("parameter pushing strategy") — ellipses option (no need to mention arguments)

struct base {
int code;
};

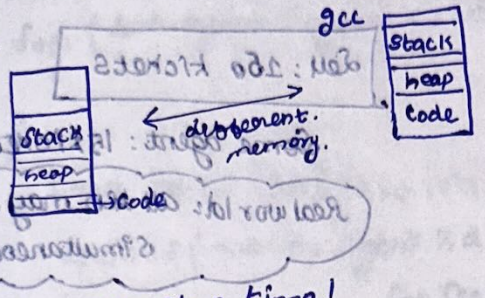
struct type_one {
int code;
}

1st field — lowest address
(useful — Networking)
IPV4, V6 [∴ mystery distance from the base address — I don't know where my 1st element is

Threading — make faster

Seemingly run — same processor?

make



FF — firefox

Clock

4 things run at a time!

In 1 processor space

1 address space (physical memory)
↓
want to host all!

OS

Memory management :

(4000)

* Build table : process & address → map to real address

* Any request to manupulate 4000 → transferred to 600000 (real add)
(proxy) manage) — virtual address → physical address space

* 40 apps — never that big-stack! (so theory possible)
* Most — app — low memory usage!
* 200 — normal (stack layer) while a billion not!

Share address spaces b/w apps — need advanced unix?

* With 1 processors — whats happens:
gcc, FF, make

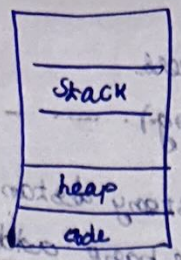
① Position eg Gcc
FF
make
Gcc
FF
make

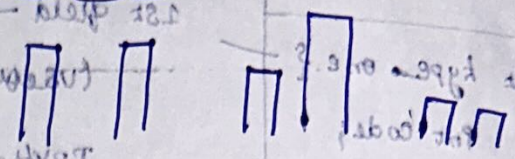
↓ Clock cycle in GHZ (extremely fast)
Looks like parallel!

*Bandwidth — needed!

*

office, Notepad.

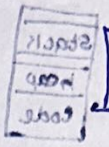




one by one — portion by portion. Looking — also time

* "Switch back & forth" — b/w threads

```
int main()
{ int numAgents = 10; int numTickets = 150;
  for (int i = 0; i < 10; i++) {
      SellTickets (i, 10);
  }
```



Sell: 150 tickets

Some agent: 15 tickets!

Real world: all ten may answer simultaneously

when particular thread exits: 15 tickets sold!

* only one at a time [Time: 10 times]

* Shedule — queue. (gates!) — pipelining!

Lecture-15 Thread library

```
int main()
{
  int numAgents = 10;
  int numTickets = 150;
  for ( int agent = 1; agent <= numAgents ; agent++){
      SellTicket (agent, numTickets/numAgents);
  }
  return 0;
}
```

```
void SellTickets (int agentID, int numTicketsToSell)
{
  while (numTicketsToSell > 0) {
      printf ("Agent %d sells a ticket.\n", agentID);
      numTicketsToSell--;
  }
  printf ("Agent %d : All done \n", agentID);
}
```

Not sequentially — I want to do it — Sametime — Simultaneously

Thread libraries: Thread: Same function running — company of other threads running!

```
int main ()
{ int numAgents = 10;
  int numTickets = 150;
  InitThreadPackage (false);
  for (int agent ; agent <= numAgents ; agent++)
  {
    Char name [32];
    Sprintf (name, "Agent %d thread", agent);
    ThreadNew (name , SellTickets , 0 , agent , numTickets / numAgent);
    }
    RunAllThreads();
    return 0;
}
```

→ must before calling library (this one)

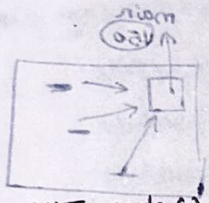
false — don't print debug prog!

sprintf → returns char array!

→ address of function!

→ Arguments

→ Thread name. [identify thread name]



Set up Thread — Then Run!

* Threads can have threads

```
if (RandomChance (0.1))
    ThreadSleep (1000);
```

→ 1 s (Stopped!) — threading

10 threads → thread 1, thread 2, ⋮, thread 3 — complete equally!

time slaves (thread 1 — do for some time ⋮)

* RandomChance(0.1) → never comes Head
* 10 levels of priority in Java.
* do for some time — wait — next thread (one thread at a time in a processor).

why threading: have concurrency

```
while ( . . . ) {
    :
    if (Random Chance (0.1))
        . . .
    :
}
```

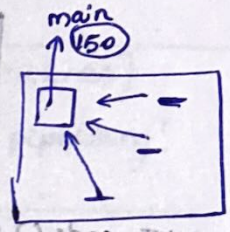
Thread library: C instructions — not atomic

a-- → takes 3 lines of asc code.

* thread library has — high priority thread — pulls off the control — every time timer goes off.

*

```
void SellTickets (int agent, int *numTicketsP)
{
    while (*numTickets > 0) {
        (*numTicketsP) --;
    }
}
```



Problem: Sold → but before *(numTicketsP)-- gone control.

Now: what do do? ; Another ticket sold...

Solution: must be completed

10 threads → sharing same source.

It doesn't manage — Disaster!

---

* Solution: Lock! when done release → next thread!

```
int main ()
{
    int numAgents = 10
    int numTickets = 150;
    Semaphore lock = SemaphoreNew (int, [redacted]);
    }
```

↳ provide atomic --.

Semaphore → non-negative int → supports atomic ++, --.

```
Semaphore wait (lock);
Semaphore Signal (lock);
```

] behind the scenes --, ++.

Semaphore wait (lock); → bell Hired!

Semaphore Signal (lock); → bell released. ++

↳ If +ve [surrounding ● - doesn't −1, it do block → on that Semaphore → so understand signalling need to be done (++)]. → I need to open door (only some one released lock)

3 arguments → idea, address of shared resource, lock.

```
int main()
{
  int numAgents = 10;
  int numTickets = 150;
  Semaphore lock = SemaphoreNew(-, ●);
                                  int
                                  → In this example alone.
                                  (two global vars can't trust each other)

  InitThreadPackage(false);
  for (int agent; agent <= numAgents; agent++) {
        char name[32];
        sprintf(name, Agent %d Thread", agent);
        ThreadNew (name, SellTickets, 3, agent, & nVmTickets,
                                                  lock);
  }
  RunAllThreads();
  return 0;
  }

void SellTickets (int agent, int *numTicketsP, Semaphore lock)
{
  while (True) {
        SemaphoreWait (lock);  ———→ −− (lockdoor)
        if (*numTicketsP == 0) break;
        *(numTicketsP) −−;
        printf ("er. ———— ");
        SemaphoreSignal (lock);  ———→ ++ (unlock)
  }
  Semaphore Signal (lock);
```

★ "deadlock" — No one able to access the source!

Semaphore ⇒ 'Sync Counter Var' >= 0

★ SW (lock) → --

★ SW (lock) → ++

SW (Sem) → 0
SS → 1
SW → 0
SW → hand the
Not allowed to progress (-)ve not allowed

Lecture-16

```
void SetTickets(int agent, int * numTicketsP,
                Semaphore lock)
{
  while(true){
    Semaphore Wait (lock);
    if (*numTicketsP == 0) break;

    (*numTicketsP) --;
    Semaphore Signal (lock);
    Printf ( — );
    //sleep
  }
  Semaphore Signal (lock);
}
```

'Concurrency' - maintained

★ Only one thread: allowed to pull through & get value into the local register & flush it out global integer. (Before next thread happen)

If I initialize: as: 0 (Instead of 1) in Semaphore New

```
while (1)
    Semaphore wait (i--)
```

→ deadend!

2 → bad (∴ No: more than 1 thread active)!

ON & OFF → 1 → in New!

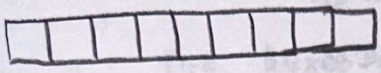
Internet model

★ one server!

★ HTML server process → Request, Serves HTML text → full rendering
↳ web page!

```
char buffer[8];

int main()
{
    IP(false);
    Thread New("writer", 0);
    Thread New("Reader", 0);
    RunAllThreads();
}
```



```
void writer()
{
    for (int i=0; i<40; i++){
        char c = prepareRandomChar();
        buffer[i%.8] = c;
    }
}

void Reader()
{
    for (int i=0; i<40; i++){
        char c = buffer[i%.8];
        processChar(c);
    }
}
```

* Reader reads: order writer writes

```
→ char buffer[8];
  Semaphore emptyBuffers(8);
  Semaphore fullBuffers(0);

void writer()
{ for (int i=0; i<40; i++){
        char c = prepareRandomChar();
        SemaphoreWait(emptyBuffers);
        buffer[i%.8] = c;
        Semaphore Signal(fullBuffers);
    }
}

void Reader()
{
    for (int i=0; i<40; i++){
        SemaphoreWait(fullBuffers);
        char c = buffer[i%.8];
        Semaphore Signal(emptyBuffers);
        processChar(c);
    }
}
```

Reader: Tries to catch writer (But never)!

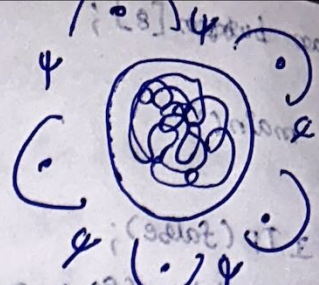
4 → doesn't change correctness, No deadlock

1 → Same ques but no! (read, write same)

0 → deadlock

8 → No dead lock!

Dining philosophy



* 5 philosophers, 5 forks
* Think, Eat (3 eat, 4 think session)
* They need - each fork - either side!
* 1 neighbour - both can't think/eat at a time

```
Semaphore forks[] = {1,1,1,1,1}

void philosopher (int i)
    for (int i=0; i<3; i++) {
        Think();

        Eat();
    }
}
```

```
Semaphore forks[] = {1,1,1,1,1};
Semaphore numAllowedToEat (2);
void philosopher (int id)
{
    for (int i=0; i<3; i++) {
        Think();
        SW (numAllowedToEat);
        SW (forks[i]);
        SW (forks[(i+1) % 5]);      // id

        Eat();
        SS (forks[i]);
        SS (forks[(i+1) % 5]);
        SS (numAllowedToEat);
    }
}
```

Lecture - 17

int DownloadSingleFiles (const char * server, const char * path);
int DownloadAllFiles (const char * server, const char * files [], int n);

```
int DownloadAllFiles(const char *Server, const char * files[], int n)
{ int totalBytes=0;
  semaphore childrenDone = 0;
  semaphore lock = 1;
  for (int i=0; i < n; i++){
      ThreadNew(__, DownloadHelper, 5, Server, files[i], & totalBytes,
                                                  lock, D);
  }for (i=0; i<n; i++) SW(childrenS one);
  Return totalBytes;
}
```

```
void DH (const char * Server, const char * path, int *numBytesP,
        semaphore lock, semaphore parentToSignal)
        int bytesDownloaded = DownloadSingleFile (Server, Path);
        SW(lock);
        (*numBytesP) += bytesDownloaded;
        SS(lock);
        SS(parentToSignal);
}
```

Lecture-18 (Ice Cream Store Simulator)

[10 customers]

[1-4 icecream cones order]

Cashier ⟷

10-40 clerks ⟷ customer

No Communication b/w cashier & clerks

manager
|
clerks [10-40]
\ /
10 customers [1-4]
|
cashier.

```
int main (__, __)
{ int totalCones=0;
  InitThreadPackage();
  SetupSemaphores();
  for (i=0; i<10; i++)
  {
    int numCones = RandomInteger(1,2);
    ThreadNew (__, customer, 1, numCones);
```

```
totalCones += numCones;
}
Thread New (__, Cashier, 0);
Thread New (__, Manager, 1, totalCones);
RunAllThread();
Free Semaphores();
return 0;
}

Struct Inspection {
        bool passed;  (false)
        Semaphore requested;  (0)
        Semaphore finished;  (0)
        Semaphore lock;  (1)
} Inspection;

void Manager (int totalConesNeeded)
{
int totalApproved = 0;
int numInspected = 0;
while (numApproved < totalNumNeeded)
{ SW (Inspection.requested);
  numInspected++;
  Inspection.passed = RandomChance (0, 1);
  if (Inspection.passed)
        numApproved++;
  SS (Inspection.finished);
}
}

void clerk ( Semaphore SendToSignal)
{
bool passed = false;
while (!passed)
Semaphore requested;  (0)
Semaphore finished;  (0)
Semaphore lock;  (1)
} Inspection
```

```
{ makeCone();
  sw (inspection.lock)
  ss (inspection.requested);
  sw (inspection.finished);
  passed = inspection.passed;
  ss (inspection.lock);
}
  ss (sendToSignal);
}
```

```
void Customer(int numCones)
{ Browse();
  Semaphore ClerksDone; 10)
  for (int i=0; i < numCones; i++)
      JN (—, Clerks, 1, ClerksDone);
  for (int i=0; i < numCones; i++)
      sw (ClerksDone);
  Semaphore Free (ClerksDone);
  WalkToCashier();
```

```
sw (line.lock)
ss (line.requested);
int place = line.number++;
ss (line.lock);
ss (line.requested);
ss (line.Customers[place]);
```

```
struct line {
    int number; (0)
    semaphore requested; (0)
    semaphore Customers [10]
    semaphore lock; (1)
}

void Cashier()
{
  for (int i=0; i<10; i++)
  {
    sw (line.requested);
    Checkout (i);
    ss (line.Customers[i]);
  }
}.
```

Lecture-19

Imperative Procedural C	OOP C++

C

Vector V;	Vector <int>
VectorNew (&V, ...);	V.push_back(4);
VectorInsert (&V, ...);	V.erase (v.begin());
VectorSort (&V, ...);	

Functional: Scheme

$f(x,y) = x^3 + y^2 + 7;$

$g(x) = f(x, x+1) + 8$

$h(x,y,z) = f(x,z) * g(x+y)$

$h(x,y,z) = f(x,z) * g(x+y)$

$= (x^3+y^2+7) * (f(x+y, x+y+1) + 8)$

```
(define Celsius -> Fahrenheit (temp)
    (+ 32 (* 1.8 temp)))
```

```
> (Celsius -> fahrenheit 100)
212
```

Shell

Kawa

>4 4	> "Hello" Hello	>#f #f

#f, #t = false, true

>#t #t	> 11.752 11.752

> 11/5 11/5	> 22/4 11/2	(Preserves type info!)

```
> (+ 1 2 3)
6
> (* (+ 4 4) (+ 5 5))
80
```

```
> (< 4 2)
#t
> (< 4 2)
#f
```

> (and (> 4 2) (< 10 5))

#t #f

#f

No struct, Classes etc..

> (car '(1 2 3 4 5))

1

Car associated with 1st node

> (cdr '(1 2 3 4 5))

(2 3 4 5) ⟶ what car doesn't

> (car (cdr (cdr '(1 2 3 4 5))))

cdr → 1

cdr → 2 3 4 5

'(1 3 (4 (5)))

(append '(2 3) 1 '(4 5)) wrong!

↓

No! (does

list(1) → Now correct.

> (cdr '(4))

() empty list → no more points

only one.

>(define add (x y)

(+ x y)) → side effect!

ADD

> (add 10 7)

17

>(add "Hi" "there")

error

Linked lists – Recursive data structures

> (sum-of '(1 2 3 4))

—

> (define sum-of (num lst)

(if (num? num lst) 0

(+ sum-of (cdr num lst))))

Lecture - 20

```
(define sum-list (num-list)
  (if (null? num-list) 0
      (+ (car num-list)
         (sum-list (cdr num-list)))))
```

```
(define fib (n)
  (if (zero? n) 0
      (if (= n 1) 1
          (+ (fib (- n 1))
             (fib (- n 2))))))
```

```
(define fib (n)
  (if (or (= n 0)
          (= n 1))
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
```

```
> (flatten '(1 2 3 4))
(1 2 3 4)
> (flatten '(1 (2 3) 4 ((5)))
(1 2 3 4 5)
```

```
> (flatten '(1 2 () 3 4)
(1 2 3 4)
```

```
(define flatten (sequence)
  (cond (((null? sequence) '())
        ((list? (car sequence))
         (append (flatten (car sequence))
                 (flatten (cdr (sequence))))
        (else (cons (car sequence)
                    (flatten (cdr sequence))))))
```

```
> (cons 4 '(1 5 7))
(4 1 5 7)
```

```
(define sorted? (num-lst)
  (or (< (length num-lst) 2)
      (and (<= (car num-lst)
               (cadr num-lst))
           (sorted? (cdr num-lst)))))

> (sorted? '(1 2 2 4 7))
#t
> (sorted? '(1 0 4 7 7 10))
#f
```

Lecture-21

```
> (double-all '(1 2 3 4))
(2 4 6 8)
> (incr-all '(1 2 3 4))
(2 3 4 5)

(define (double x) (* x 2))
(define (incr-all x) (+ x 1))
```

```
> (map incr '(1 2 3 4))
(2 3 4 5)
```

— Racket kawa and scheme!

```
(define (my-unary-map fn seg)
  (if (null? seg) '()
      (cons (fn (car seg))
            (m-u-m fn (cdr seg)))))
```

```
(define (average num-lst)
  (/ (apply + num-lst)
     (length num-lst)))
```

```
(define (flatten seg)
  (if (not (list? seg) (list seg)
      (apply append
             (map flatten seg))))
```

lambda – anonymous function!

```
(define (translate seg delta)
  (define (shift-by x)
    (+ x delta))
  (map shift by seg))
```

```
(define sum
  (lambda (x y) (+ x y)))
```

Lecture – 22

Power Set:

(1 2 3)

⇓

(() (2) (3) (2 3)
(1) (1 2) (1 3) (1 2 3))

$\boxed{\text{Powerset – all subsets}}$

```
(define (ps set)
  (if (null? set) '(())
      (append (ps (cdr set))
              (map (lambda (subset)
                     (cons (car set) subset))
                   (ps (cdr set))))))
```

```
(define (ps set)
  (if (null? set) '(())
      (let ((ps-rest (ps (cdr set))))
        (append ps-rest
                (map (lambda (subset)
                       (cons (car set) subset))
                     ps-rest)))))
```

```
(let ((x exp v1)
      (y exp v2)
      (z exp v3))
```

```
(let ((x —)
      (y — ,))
      (a x y))
```

```
lambda (x y)
(a x y)) — , — )
```

```
>(permute '(1 2 3))
((1 2 3) (1 3 2) (2 1 3) (2 3 1) (3 1 2)
 (3 2 1))
```

```
(define (permute items)
  (if (null? items) '(())
      (apply append
             (map (lambda (elem)
                    (map (lambda (permutation)
                           (cons elem permutation))
                         (permute (remove elem items))))
                  items))))
```

Lecture - 23

```
>(map car '((1 2) (3 4) (5 6 7)))
(1 3 5)
> (map + '(1 2) '(10 20) '(100 400))
(111 422)
>
```

```
(define (unary-map fn seq)
  (if (null? seq) ()
      (cons (fn (car seq))
            (unary-map fn
                       (cdr seq)))))
```

(define (bar x a) b c . d) → (other c (list))

(bar 1 2 3 4 5)

a, b, c '(4 5 6)

```
(define (map fn . first-list other-lists)
  (if (null? first-list) '()
      (cons (apply fn
                   (car first-list)
                   (map car other-lists))
            (map-many- map car
                       other-lists)))
```

Scheme
ML
Haskell
Erlang

Lecture-24 - Python!

Python - scripting language!, imperative, object oriented, functional!
↳ dynamic

C, C++ → Not dynamic.

"hello". Starts with ('he') → No need for object to surround object cons that.

"hello there". capitalize()

.lstrip()

Lecture-25 → Python!

Lecture-26

XML, Parsing.

LISP/Scheme - 1959
ML/OCaml (1979)
Miranda - 1985
Haskell - 98 (2003)

---

int array[5]

access → array[6]
array[7]
array[8]

Sooner Core dump (Not in boundary).

→ 787 – Boeing – Plane – on – greater than 8 months – turn off (electric)

Found out : Bug : Counter : overflow! (-ve)!

→ Linux machine : Anything (Backspace – 28 times) – Unlocked!

→ Reason : Bounds error!

→ Boot loader : (Some bug)

Array are not pointers – can't give other addresses!

Dynamic prog.

Minimum spanning tree

Graph traversal → Depth first search (DFS)

↳ Breadth – First " (BFS)

* Borůvka
* Kruskal
* prim

Shortest path :

* Bellmann – Ford
* Dijkstra
* Floyd – warshall.

$\boxed{\text{Strstr}}$

→ char * Strstr (const char *S1, const char *S2);

↓ mainstring ↓ substring

* Returns first character found! (else empty string → null pointer)

Say : S_1 → Some string $\xrightarrow{\text{returns}}$ S1!

S2 → empty string

---

p = Strstr (S1, S2);

S1 [] = "GeeksforGeeks"

S2[] = "for"

print p → forGeeks.

S1 = "Fun with STL";

S2 = "STL";

char *P = Strstr (S1, S2); print → Fun with Strings

Strcpy (P, "Strings"); S1

Variadic functions

<stdarg.h>

int funcName (data_type variable_name, ...)

int printf (const char *format, ...);

va_start (va_list ap, argN) → enables access to variadic function

va_arg (va_list ap, type) → accesses next variadic func arg.

va_copy (va_list dest, va_list src) → makes a copy of the variadic func arguments

va_end (va_list ap) → ends traversal of the variadic func. arg.

va_list holds → info needed by start, arg, copy, end.

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

int AddNumbers (int n, ...)
{
    int sum = 0, i; va_list ptr;
    //Declare pointer to the list pointer
    va_start (ptr, n);
    for (i=0; i<n; i++)
    {
        sum += va_arg (ptr, int);
    }
    va_end (ptr);
    return sum;
}
```

start, get arg!, end

```
int sum (int count, ...)
{ int sum=0;
    va_list ptr;                    // initialize variable list.
    va_start (ptr, count);
    for (int i=0; i< count; i++)
        sum += va_arg (ptr, int);   // get argument
    va_end (ptr);                   //end va_list pointer.
    return sum;
```

```
int min (int Count, ...) {
    int min = 100000000, temp;
    va_list ptr;
    va_start (ptr, Count);
    for (int i = 0; i < Count; i++)
    {
        temp = va_arg(ptr, int);
        if (min > temp)
            min = temp;
    }
    va_end (ptr);
    return min;
}

int max (int Count, ...) {
    int max = -100000000, temp;
    va_list ptr;
    va_start (ptr, Count);
    for (int i = 0; i < Count; i++)
    {
        temp = va_arg(ptr, int);
        if (max < temp);
            max = temp;
    }
    va_end (ptr);
    return min;
}
```

---

strlen(str)
strcat (str1, str2)
strcpy (")
strncat (str1, 2, n)
strncpy (str1, str2, n) ; → copies n deg of str2 to str1
strncmp (str1, str2)
strncmp (str1, str2, n)
strchr (str1, c)
strrchr (str1, c) → search in reverse
str... (str1, str2) → pointer to str2 (substring)

atoi (str) → int

atof (str) → float

atol (str) → long int

int (*op [4]) (int, int)

↳ Function pointer array of 4 taking int, returning int.

```
typedef struct {
    int x;
    int y;
} point;

typedef struct {
    float radius;
    point center;
} circle;
```

```
Circle c;

c.center.y;
```

Pointers to struct

```
struct myStruct *struct_ptr;
struct_ptr = &struct_var;
struct_ptr -> struct_mem;
```

dereference then ask member.

---

math.h
- EDOM → domain out of range
- ERANGE → RANGE error

* __DATE__ → Current date as a string. Mm dd yyyy

__TIME__ → Current time " hh:mm:ss

__FILE__ Current filename as a string

__LINE__ Current line number as int

__STDC__ 1