Programming in C
Week 4

23.9.2015

Tiina Niklander

NO Lecture meeting on Wednesday 30.9.

Week3 – Lecture questions

”segmentation error” – What to do?

- use extra print statements
- avoid using netbeans and operate on the command line directly – compile for gdb and use it to locate the problem
- use valgrind to evaluate your program
- put assert macros to check your assumptions about the behavior

Week 3: Lecture questions

More clarity for the task definitions, please!

- There have been some improvements based on the feedback, but difficult to figure out in advance what term selection may cause problems due different understanding of it.

- Solution: Give very specific feeadback (week, task number, unclear sentence, - your understanding of it) – they can be correct as soon as the specific information reaches the teachers

Week 3: Lecture questions

NetBeans problems in windows:

- Aalto course material and their wiki has more information about possible problems, they may have hints on solving this

- Problems on department’s computers with NetBeans: Discuss with paja instructors and if not solved send a very detailed description of the problem, computer name, date and time as error report to either department’s IT personnel or me.

Week3: Lecture questions

- String handling, passing string as an argument that was received as a char*
- Not always very clear to do m / *m / &m

- Both of these are related to C being low-level language and requiring the programmer to understand the computer architecture aspects related to memory, memory addressing, and heap and stack of a process.

- HINT: Remember to reflect back to CompOrg 1 material

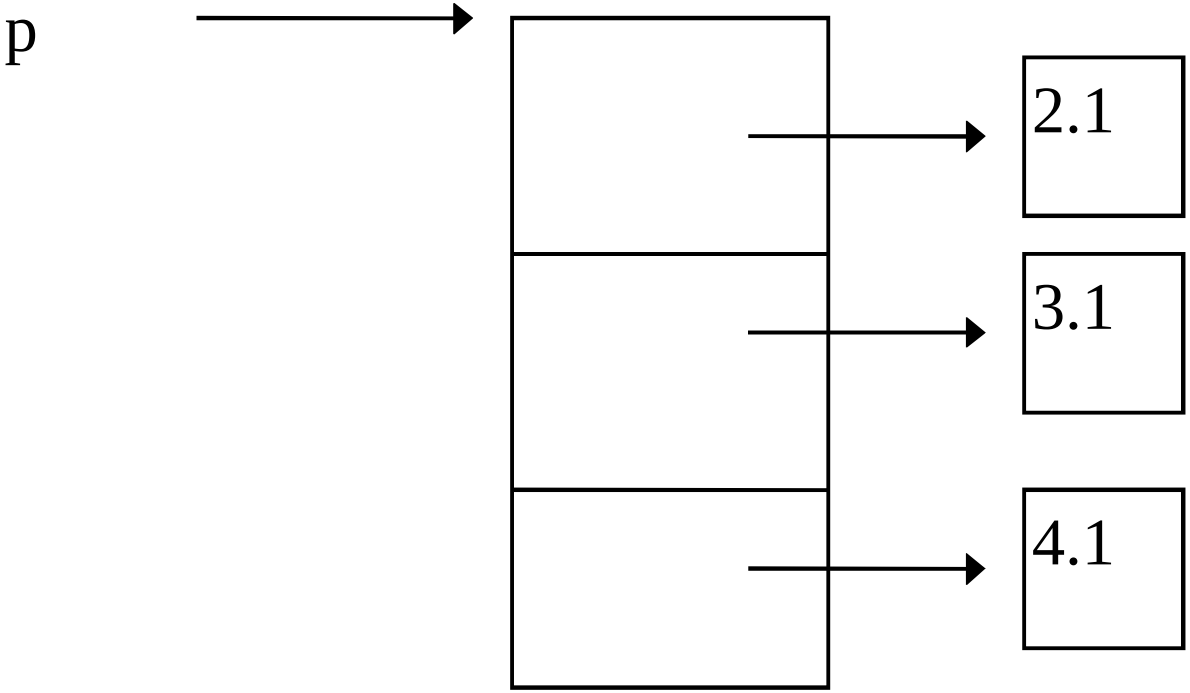
Week3: Questions and answers

- Files
- File open
- feof

- program logic

- OTHER ISSUES?

Pointer to array of pointers



Arrays of pointers to individual double elements

Dereferencing has to be done twice   **p

Allocating space to array and the referenced elements

```
double **block;
#define SIZE 3
if((block=calloc(SIZE, sizeof(double*)))==NULL)
     error;

for(i = 0; i < SIZE; i++)
  if((block[i]=calloc(1, sizeof(double)))==NULL)
          error;

*(*block) = 2.1;
block[0][0] = 2.1;
```

Allocation for the array

Allocation for one element at a time

Referencing the element and freeing them

Setting the values to the referenced elements

```
for(i = 0; i < SIZE; i++)
      block[i][0] = 2.1 + i;
```

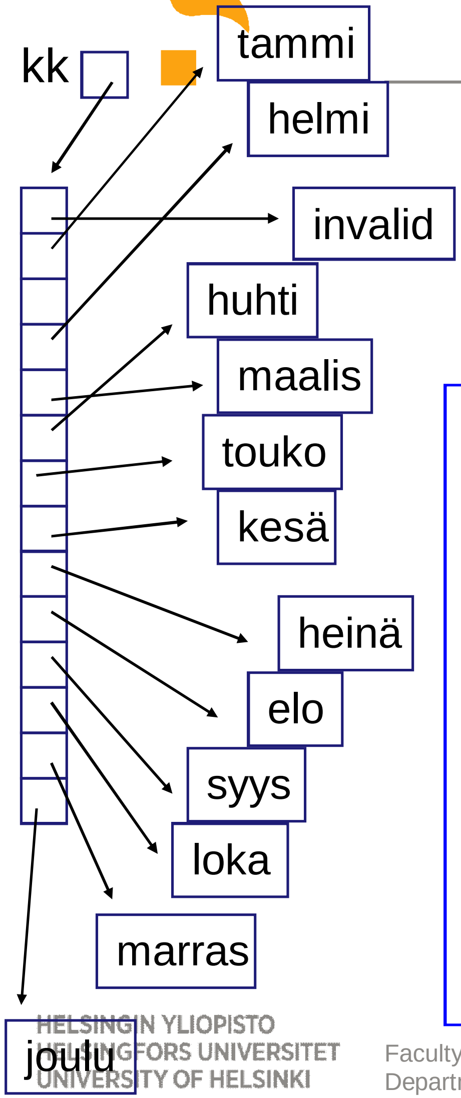
Freeing the memory: referenced elements and pointer array

```
for(i = 0; i < SIZE; i++)
      free(block[i]);
   free(block);
   block = NULL;
```

First elements

..and then array

Pointer arrays: array of strings



String arrays are always pointer arrays, since each string is accessed via a pointer.

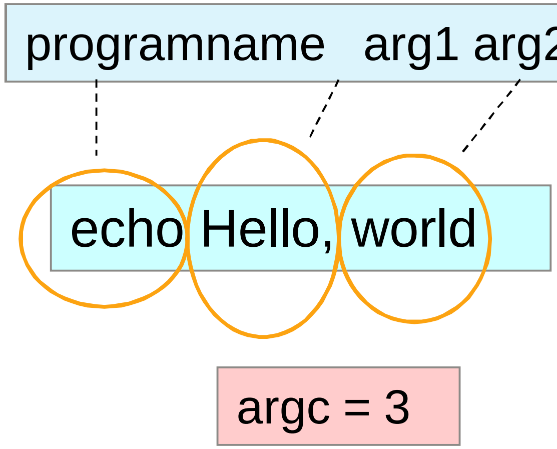
```
char *monthFinnish(int k)
{
  static char *kk[] = {”invalid”, ”tammi”, ”helmi”,
maalis”, ”huhti”, ”touko”, ”kesä”, ”heinä”, ”elo”,
”syys”, ”loka”, ”marras”, ”joulu”};

return ( (k<1 || k > 12)  ? kk[0] : kk[k] );
}
```

Command line arguments

```
int main (int argc, char **argv);
int main (int argc, char *argv[]);
```

argc count of strings

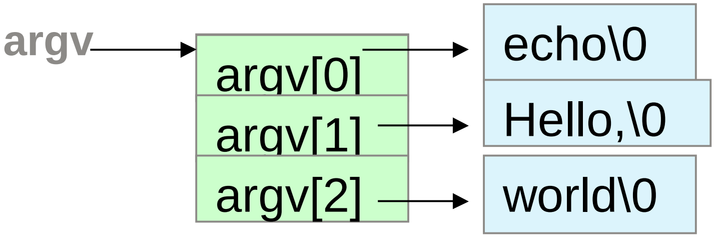
argv pointer to string array



programname arg1 arg2 …

echo Hello, world

argc = 3



argv

argv[0] → echo\0

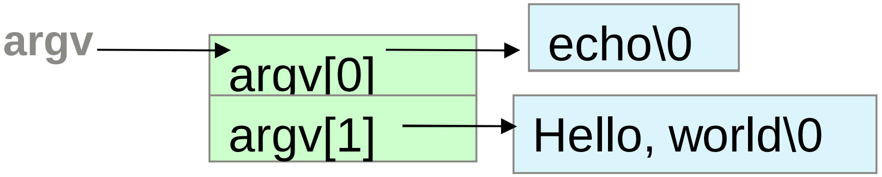
argv[1] → Hello,\0

argv[2] → world\0

echo ”Hello, world”

argc = 2

” ” –work in some systems!



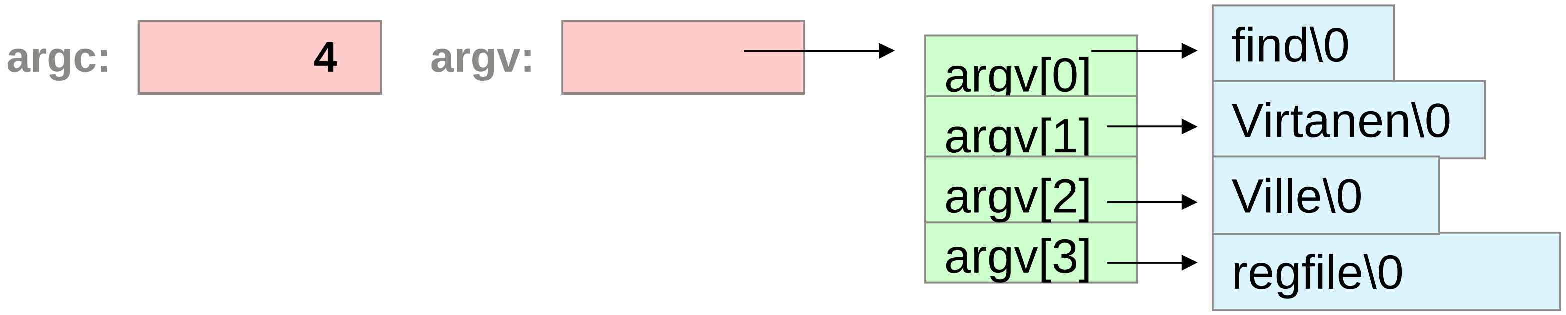
argv

argv[0] → echo\0

argv[1] → Hello, world\0

Using comm.line.args

find Virtanen Ville regfile



argc: 4

argv:

argv[0]

argv[1]

argv[2]

argv[3]

find\0

Virtanen\0

Ville\0

regfile\0

argv[0] or argv   contains 1.arg, that is the program name (”find”),

argv[1] or argv+1 contains 2. argument (”Virtanen”)

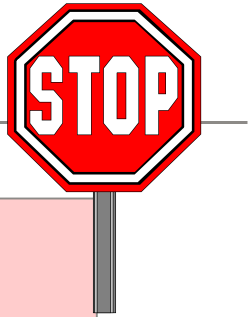
argv[2] or argv+2 contains 3. argument (”Ville”)

argv[3] or argv+3 contains 4. argument (”regfile”)

argv[0][0] or (*argv)[0] or **argv is the first character of the first arg (’f’)

argv[2][4] or (*(argv+2))[4)] or *(*(argv+2)+4) is fifth char of third arg.

Checking the number of arguments



```
/* Check that count is correct*/
int main(int argc, char **argv) {
 ….
   switch(argc) {
   case  4: …  /* all information on command line*/
   case 3:  …  /*OK!  Use the preset file name*/
   default: fprintf(stderr, ”Incorrect usage: %s .. \n",
               argv[0]);  /*Would be better to inform correct usage*/
               return EXIT_FAILURE;
}
```

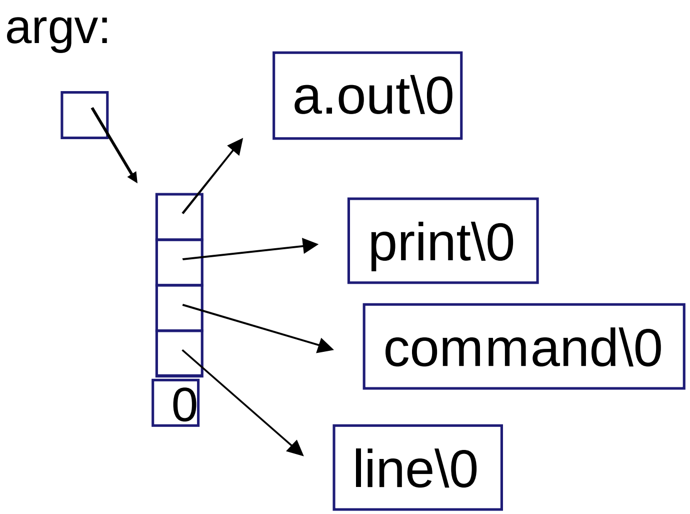
Example program
(from first week’s slide set)

- What does this program do?

```
#include <stdio.h>
/*   Explaining comment removed */
int main(int argc, char** argv)
{
  int i;

  for (i=0; i < argc; i++)
     printf(”%s%s”, argv[i],
          (i <argc-1) ? ” ” : ””);
   printf(”\n”);
return 0;
}
```

Example program:
a.out print command line

```
#include <stdio.h>
/* Echo the command line with params */
int main(int argc, char** argv)
{
  int i;

  for (i=0; i < argc; i++)
     printf(”%s%s, argv[i],
          (i <argc-1) ? ” ” : ””);
   printf(”\n”);
return 0;
}
```

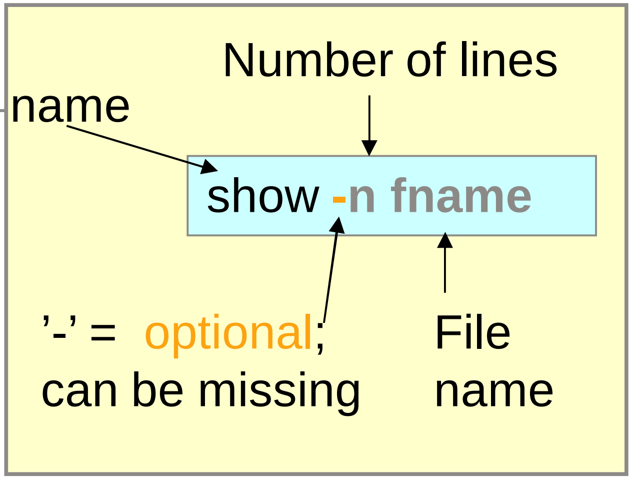


NOTICE:
- Parameters
- Array indexing

Modification:
How would you avoid printing the name of the program?

Using command line arguments:

Changing program behaviour with options ’-x’



Number of lines

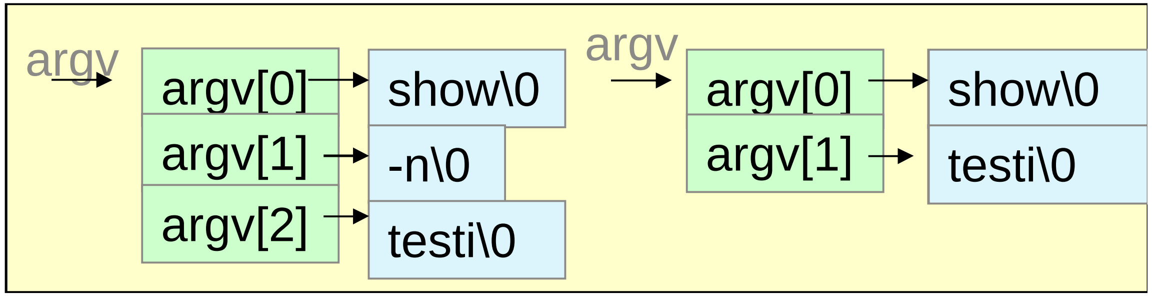
name

show -n fname

’-’ = optional; can be missing

File name

```
#define DEFAULT 10
#define MAX      80
/*Prints the n lines of the file to std*/
int display(const char *fname, int n,  int Max);
int main(int argc, char **argv) {
    int lines = DEFAULT;
    switch(argc) {
    case 3: /* selvitä rivien lukumäärä argumentti */
        if(argv[1][0] != '-' ||  sscanf(argv[1] + 1, "%d", &lines)!=1 || lines <= 0)
        return EXIT_FAILURE;
        argv++;          /* no break: retrieve filename */
    case 2:  if(display(argv[1], lines, MAX) == 0)  return EXIT_FAILURE;
               break;
    default:  return EXIT_FAILURE;
 }
 return EXIT_SUCCESS;
}
```



argv

argv[0] show\0

argv[1] -n\0

argv[2] testi\0

argv

argv[0] show\0

argv[1] testi\0

Command line parameters: typical usage - options

```
static int process_parameters(int argc, char *argv[]) {
  int i, string_found =0;
  for(i=1; i<argc; i++){ /* process command switches. Note side effects! */
    if (argv[i][0] == '-') {  switch (argv[i][1]) {
      case 'c': count_lines = TRUE; break;
      case 'i': ignore_case = TRUE; break;
      case 'b’: line_beginning = TRUE; break;
      default: printf("Unknown option %s - ignored \n", argv[i]);  break;
    }} else {
      if (!string_found) {
        copy(string, argv[i], STRINGSIZE); string_found =1;
      } else {
        printf("Only one search string! \n"); return FALSE;   } }}
  if (!string_found) {
    printf("The search string must be given!\n"); return FALSE; }
  return TRUE; }
```

options: -c, -i, ja -b

Search string cannot start with character -

Functions always return value!

Function pointers

Function pointers

- Functions also have an address and we can use that address as a value of a function pointer.

```
int (*lfptr) (char[], int);
lfptr = getline;  /* when int getline(char s[], int len); */
```

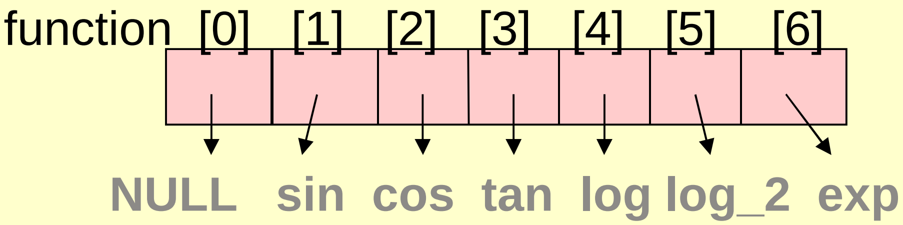
- Function pointers can be
  - passed to other functions, returned from functions
  - stored in arrays,
  - assigned to other function pointers
- stdlib.h has function qsort, whose one argument is the sorting function

Function array from include/linux/quota.h

```
 /* Operations which must be implemented by each quota format */
struct quota_format_ops {
         int (*check_quota_file)(struct super_block *sb, int type);
               /* Detect whether file is in our format */
        int (*read_file_info)(struct super_block *sb, int type);
               /* Read main info about file - called on quotaon() */
        int (*write_file_info)(struct super_block *sb, int type);
               /* Write main info about file */
       int (*free_file_info)(struct super_block *sb, int type);
                /* Called on quotaoff() */
       int (*read_dqblk)(struct dquot *dquot);
               /* Read structure for one user */
        int (*commit_dqblk)(struct dquot *dquot);
               /* Write structure for one user */
       int (*release_dqblk)(struct dquot *dquot);
               /* Called when last reference to dquot is being dropped */
 };
```

```
typedef  double (*funcptr) (double );
```

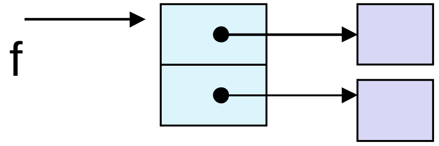
```
void main (void) {
int choice;  double x, fx;
funcptr fp;
………..
funcprt function[7] = {NULL, sin, cos, tan, log , log_2, exp}; /*defined functions*/
/*  print the function menu, for the use to make a selection*/
 ….
scanf (”%i”, &choice);
/* check that the user given value is valid*/
…
if (choice ==0) break;
printf(”Enter x: ”); scanf(”%lg”, &x);
fp = function[choice];
fx = fp(x);
printf(”\n  (%g) = %g\n”, x, fx);
}
}
```



Complexity of expressions?

[] has higher preceedence than *

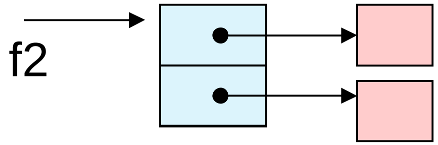
double *f[2];



f

Double variables

double (*f2[2])()



f2

Functions returning double value

double (*f3())[]

f3 – function returns a pointer to array of doubles

double *(f4[])()

INCORRECT! Cannot have the function array, only individual function addresses.

Java: ”overriding”

Example: function pointer as argument

Function that can change the sort algorithms during the execution based on number of elements

```
int (*fp) (void);
```

Function pointer

```
int *fp()
```

Function returns pointer to int!

```
int fname(); /* function must have same prototype */

fp = fname;  /*  fp() means now same function as fname()
```

```
void qsort(*line[], int left, int right, int (*comp)(void *, void*));
```

Function search

```
/* Search a block of double values */
int search( const double *block , size_t size,
        double value) {
  double *p;

  if(block == NULL)
    return 0;

  for(p = block; p < block+size; p++)
    if(*p == value)
      return 1;

  return 0;
}
```

Pointer as
call by value

Go through the
structure

Generic function search

C has no polymorfism, but we can emulate it with generic pointers (of type void*).

Function prototype can have all arguments and return value of generic (undefined) type void

```
int searchGen(const void *block,
              size_t size, void *value);
  /* Generic type is not enough */
  /* Must have more information and arguments */

int searchGen(const void *block,
  size_t size, void *value, size_t elSize
  int (*compare)(const void *, const void *));
```

Prototype:

structure

Size of element

Number of elements

Comparison function

Call back function

Calling routine must define a **Call back** function

Using typed arguments in the call back function prototype

```
int comp(const double *x, const double *y) {
   return *x == *y;
}
```

With undefined arguments the prototype must also use undefined arguments

```
int comp(const void *x, const void *y) {
   return *(double*)x == *(double*)y;
}
```

Generic search - calling routine

```
/* Application of a generic search */
#define SIZE 10
double *b;    double v = 123.6;     int i;
int main (void) {
 if(MALLOC(b, double, SIZE))
   exit(EXIT_FAILURE);
 for(i = 0; i < SIZE; i++) /* initialize */
   if(scanf("%lf", &b[i]) != 1) {
        free(b);
        exit(EXIT_FAILURE);
   }
 printf("%f was %s one of the values\n",
   v, searchGen(b, SIZE, &v, sizeof(double), comp)
              == 1 ? "" : "not");
return 0;  /* tai exit(EXIT_SUCCESS); */
}
```

Generic search function

```
int searchGen(const void *block,
  size_t size, void *value, size_t elSize,
  int (*compare)(const void *, const void *)) {
  void *p;
  if(block == NULL)
    return 0;
  for(p = (void*)block; p< block+size*elsize;
       p = p+elsize)
    if(compare(p, value))
      return 1;
  return 0;
}
```

NOTE: Pointer operations must use the size of the element!

Multidimensional arrays

Multidimensional arrays

Multidimensional arrays in C are actually single dimensional arrays with element as arrays

int t[3][2] = { {1,2},{11,12}, {21,22}};

	0	1
0	1	2
1	11	12
2	21	22

```
for (i=0; i<3; i++) {
   for (j = 0; j<2; j++)
     printf (”t[%d][%d] = %d\”,i,j t[i][j]);
  putchar(’\n’);
}
```

```
t[0][0] = 1      t[0][1] = 2
t[1][0] = 11    t[1][1] = 12
t[2][0] = 21     t[2][1] = 22
```

```
static char days [2][13] ={
 {0, 31, 28, 31, 30, 31,30,31, 30, 31,30, 31},
 {0, 31, 29, 31, 30, 31,30,31, 30, 31,30, 31}
 };

count = days[leap][2];
```

When leap ==0, then count = 28,
when leap ==1, then count = 29

Week 5 topics

NO MEETING ON WED 30.9.!!!!

Struct

```
struct info {
        char firstName[20];
        char lastName[20];
        int age;
 };
 struct info i1, i2;
```

```
typedef struct InfoT {
        char firstName[20];
        char lastName[20];
        int age;
 } InfoT;

 InfoT p1;
```

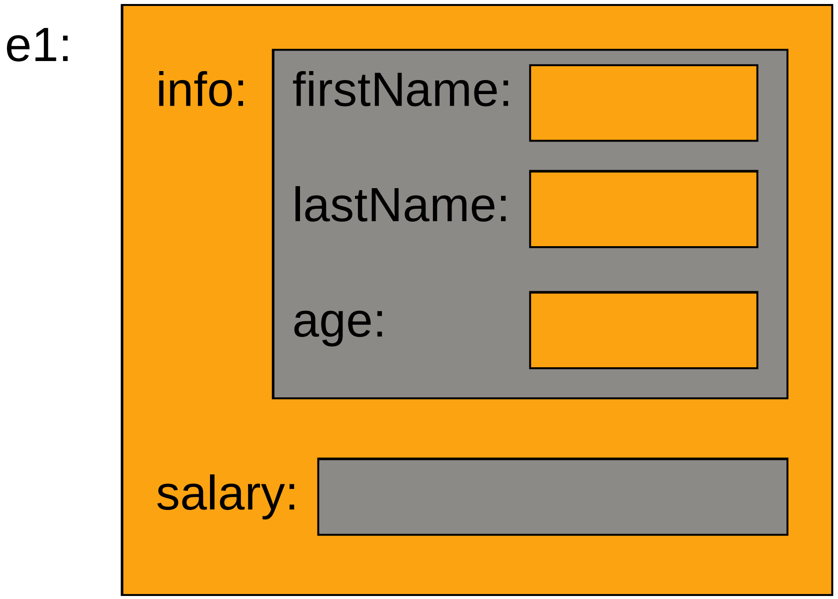
Preferable alternative!

Access to struct field with ‘.’ notation struct.field:

```
    p1.age = 18;
  printf("%s\n", i2.firstName);
```

```
struct info {
        char firstName[20];
        char lastName[20];
        int age;
 } k1, k2;
```

Struct within struct



```
typedef struct {
      char firstName[20];
      char lastName[20];
      int age;
 } InfoT;
 typedef struct {
      InfoT info;
      double salary;
 } EmployeeT;
 EmployeeT e1;
```

```
e1.info.age = 21;
e1.salary = 125.6;
```

Pointer to struct

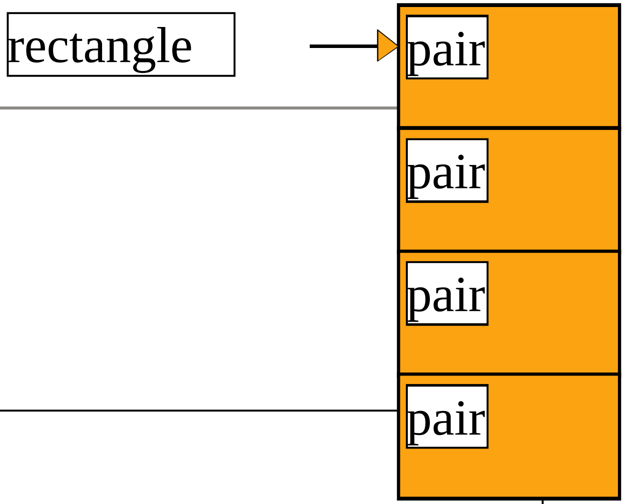
```
typedef struct pair {
      double x;
      double y;
 } PairT, *PairTP;
 PairT x;
 PairTP p;
```

Access to struct field:

(*p).x or **p->x**

```
PairT w;
PairTP q;
PairTP p = &w;

if((q = malloc(sizeof(PairT))) == NULL) …
if((q = malloc(sizeof(struct pair))) == NULL) …
   w.x = 2;
   p->x = 1;      (*p).x = 1;       ~~*p.x = 1;~~
   q->y = 3.5;
```

Array of structs



Array items can be of any type.

Access to stuct fields as with individual structs.

```
PairTP rectangle;
PairTP aux;
double x, y;

if((rectangle= malloc(4*sizeof(PairT)))==NULL)error;
 for(aux = rectangle; aux < rectangle + 4; aux++) {
   printf("Enter two double values:");
   if(scanf("%lf%lf", &x, &y) != 2) /* error */
     break;
   constructorP(aux, x, y);
 }
```

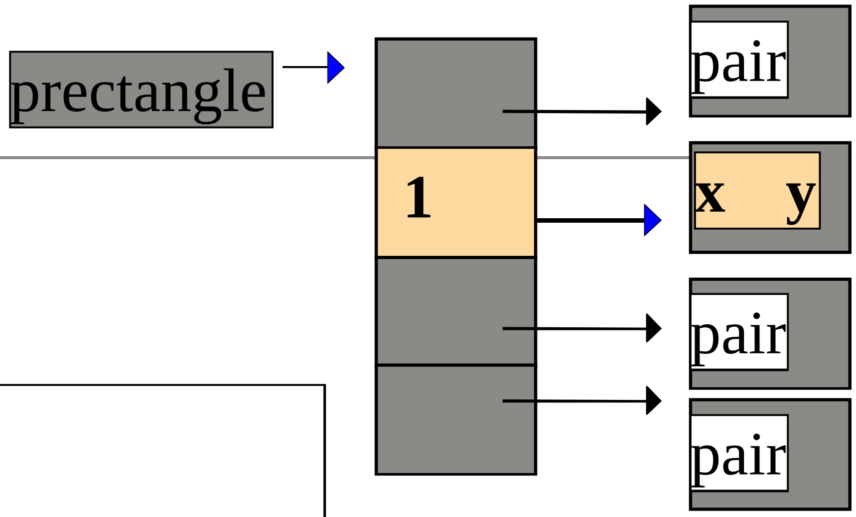
Array of structs

Access to fields:

prectangle[1][0].x

prectangle[1]->x

(prectangle+1)->x



```
int i;
PairTP *prectangle;
for(i = 0; i < 4; i++) {
   printf("Enter two double values:");
   if(scanf("%lf%lf", &x, &y) != 2)
      error;
   if((prectangle[i] = constructor(x, y))
        == NULL)
      error;
}
for(i = 0; i < 4; i++)
  printf("vertex %d = (%f %f)\n", i,
    prectangle[i][0].x, prectangle[i][0].y);
```

enum – enumerated type

Enumerated constants usually represent integer values 0,1,2,…

Can start from different value than 0.

```
typedef enum opcodes {
   lvalue, rvalue,
   push, plus
} OpcodesT;

enum opcodes e;
OpcodesT f;

int i = (int)rvalue; /*i=1*/
```

```
enum opcodes {
   lvalue = 1, rvalue,
   push, plus
};

enum opcodes e = lvalue;
if(e == push) …

int i = (int)rvalue;/*i=2*/
```

Enum as return value

Using enum as return value from a function.

Error messages in a string table, enum value used to index the array.

```
typedef enum {
  FOPEN, FCLOSE, FOK
} FoperT;

#define TOINT(f)   ((int)(f))

char *Messages[] = {
     "File can not be opened",
     "File can not be closed",
     "Successful operation",
     "This can not happen"
};
```

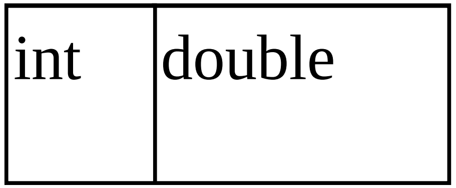
```
FoperT process();

printf("result of calling process() is %s\n",
   Messages[TOINT(process())];
```

union

Struct

Fields continuous.

```
struct intAndDouble {
  int i;
  double d;
  };
```

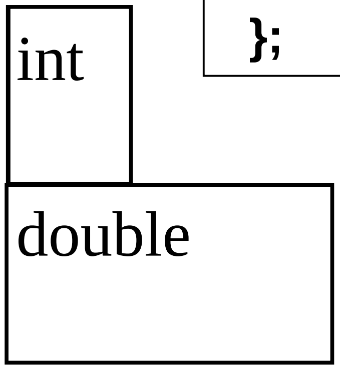


intAndDouble

Union

Fields overlapping

```
union intOrDouble {
  int i;
  double d;
  };
```



intOrDouble

union - usage?

Usually as part of struct

Special tag field indicate how to interprete the union

Used in communication protocols to save space

Reference to union fields using the point '.' notation

```
typedef enum {
  integer, real
} TagTypeT;

typedef struct {
  TagTypeT tag;
  union {
    int i;
    double d;
  } value;
} TaggedValueT;
TaggedValueT v;

if(v.tag == integer)
  …v.value.i…;
else
  …v.value.d…;
```

Bitwise operations:

&	bitwise and
|	bitwise or
^	bitwise xor (exclusive or)
<<	left shift
>>	right shift
~	one's complement

```
#include <stdio.h>
/* Bittipeliä*/
int main(void)
{
  enum {LL = 011 };
  int i, j;

  i = 0;
  j = i | LL;
  printf(”i: %d, LL (okt):%o, i|LL: %d, oktaalina %o\n”,
             i, LL, j, j);
  printf(”1 & 6: %d, 1 && 6: %d\n”,
             1 & 6, 1 && 6);
  printf(”1<<3: %d, 8>>3: %d\n”,
             1<<3, 8>>3);
return 0;
}
```

NOTE: && logical AND

Prints:

```
i: 0, LL (okt):11, i|LL: 9, oktaalina 11
1 & 6: 0, 1 && 6: 1
1<<3: 8, 8>>3: 1
```