CPROUC/2021-2022

Jesse op den Brouw

CPROUC

Pointers

DE HAAGSE HOGESCHOOL

Pointers

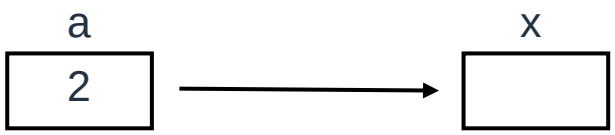
- Een variabele ligt in het geheugen.
- De computer (compiler) kan deze geheugenplaats schrijven:

```
int a = 2; /* op geheugenplaats van a wordt 2 gezet */
```

- De computer (compiler) kan deze geheugenplaats ook lezen:

```
int x = a; /* de inhoud van a wordt in x geplaatst */
```

- In het geheugen:



DE HAAGSE HOGESCHOOL

Pointers

- Er is nog een ander soort variabele: de *pointer*.
- Een pointer is een variabele met als inhoud het *geheugenadres* van een (andere) variabele.
- We definiëren een pointer als volgt:

```
int *pi; /* pi is een pointer naar een int */
```

Pointers – definitie

- We definiëren een pointer als volgt:

```
int *pi; /* pi is een pointer naar een int */
```

- We kunnen vervolgens de pointer laten *wijzen* naar een variabele:

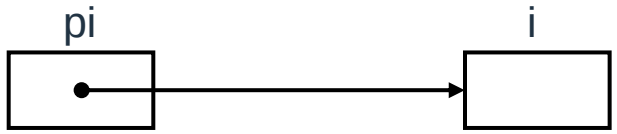
```
int i;   /* de integer */
int *pi; /* pi is een pointer naar een int */
pi = &i; /* pi wijst naar i */
```

Pointers – definitie & gebruik

- We kunnen vervolgens de pointer laten *wijzen* naar een variabele:

```
int i;   /* de integer */
int *pi; /* pi is een pointer naar een int */
pi = &i; /* pi wijst naar i */
```

- Uitbeelding:



Pointers – naar type

- Een pointer moet wijzen naar een type variabele:

```
char c, *pc;    /* een character */
int i, *pi;     /* de integer */
float f, *pf;   /* een float */
double d, *pd; /* een double */
```

- Een pointer mag ook wijzen naar een array, een structure en een array van structures.

Pointers – benaderen

- De integer en de pointer:

```
int i, *pi;     /* de integer en pointer */
pi = &i;        /* pi wijst naar i */
```

- Via de pointer kunnen we een variabele benaderen:

```
*pi  = 2;       /* zet variabele i op 2 */
```

Pointers – benaderen

- Via de pointer kunnen we een variabele benaderen.
- Verhoog variabele i met 1:

```
*pi = *pi + 1;
```

- Druk variabele af:

```
printf("%d", *pi);
```

Pointers – NULL-pointer

- Een pointer kan geïnitialiseerd worden met `NULL`:

```
#include <stdio.h>
int *pi = NULL;
```

- De pointer wijst nu naar “niets”.
- Een NULL-pointer kan niet gebruikt worden:

```
*pi = 1;    /* Oops! */
```

Verwisselen (foutief)

- Functies krijgen (tot nu toe) gegevens mee volgens *Call by Value*.
- Verwisselen van gegevens kan dan niet.
- In veel sorteerprogramma's komt het verwisselen van gegevens voor:

```
void wissel(int a, int b) {

  int hulpje = a;
  a = b;
  b = hulpje;
}
```

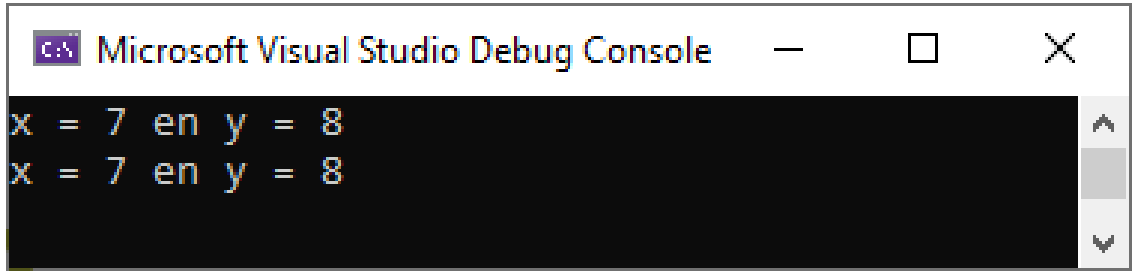
Verwisselen (foutief)

- Als we deze functie als volgt gebruiken:

```
int main(void) {

    int x = 7, y = 8;

    printf("x = %d en y = %d\n", x, y);
    wissel(x, y);
    printf("x = %d en y = %d\n", x, y);
    return 0;
}
```

Verwisselen (foutief)

- Dan zien we dat de gegevens niet verwisseld zijn:



- x en y zijn niet verwisseld!
- Dat kan ook niet want de functie krijgt *kopieën* mee van x en y!
- Dit probleem kan worden opgelost met behulp van *pointers.*

Verwisselen – correct

- We definiëren de functie als volgt:

```
void wisselp(int *pa, int *pb) {

    int hulpje = *pa;
    *pa = *pb;
    *pb = hulpje;
}
```

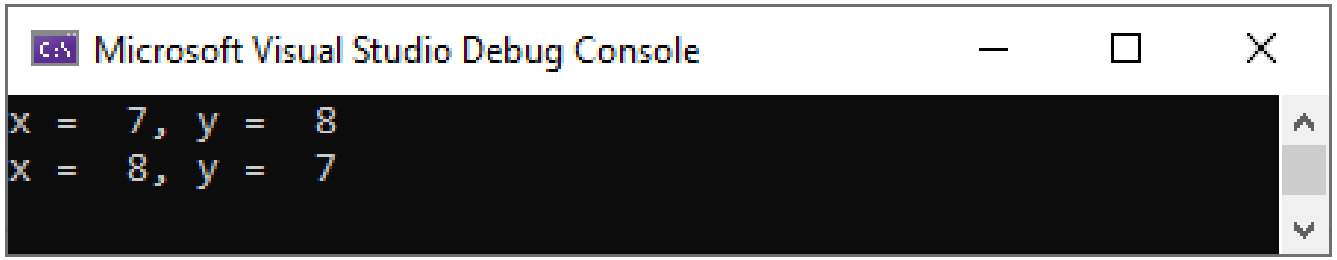
Verwisselen – correct

- En roepen de functie als volgt aan:

```
int main(void) {

   int x = 7, y = 8;

   printf("x = %d en y = %d\n", x, y);
   wisselp(&x, &y);
   printf("x = %d en y = %d\n", x, y);
   return 0;
}
```

Verwisselen – correct

- Dan zien we dat de gegevens wel verwisseld zijn:



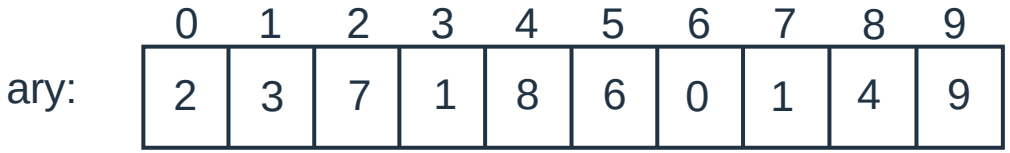
- Dat komt omdat niet de gegevens maar een pointer naar de gegevens zijn meegegeven!

Arrays

- Een array is een samengesteld datatype.

```
int ary[] = { 2, 3, 7, 1, 8, 6, 0, 1, 4, 9 };
```

- In het geheugen:

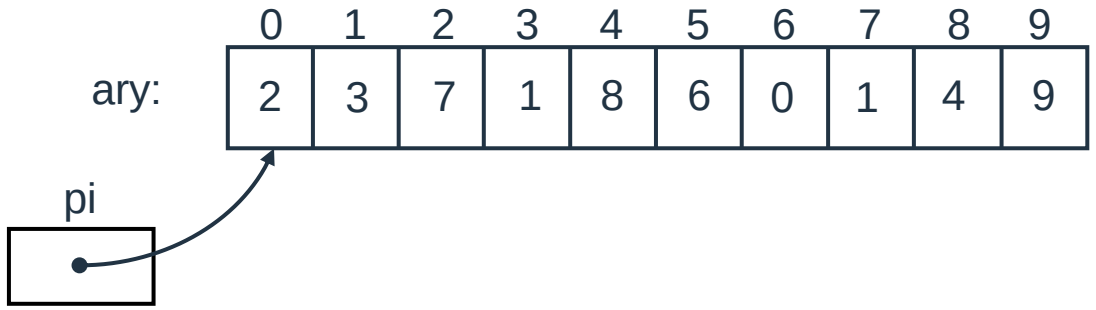


- Array-element kan benaderd worden met `ary[x]` (x=0 t/m 9).

Arrays – met pointers

- Via een pointer:

```
int ary[] = { 2, 3, 7, 1, 8, 6, 0, 1, 4, 9 };
int *pi;

pi = &ary[0];
```



- Of korter:

```
pi = ary; /* naam van array is adres van 1e element */
```

Arrays – met pointers

- Benaderen van array via:

```
int i;
i = *pi;
```

- Maar ook mag:

```
i = pi[0];
```

- Arrays en pointers zijn uitwisselbaar!

Pointers – rekenen

- Benaderen van array via:

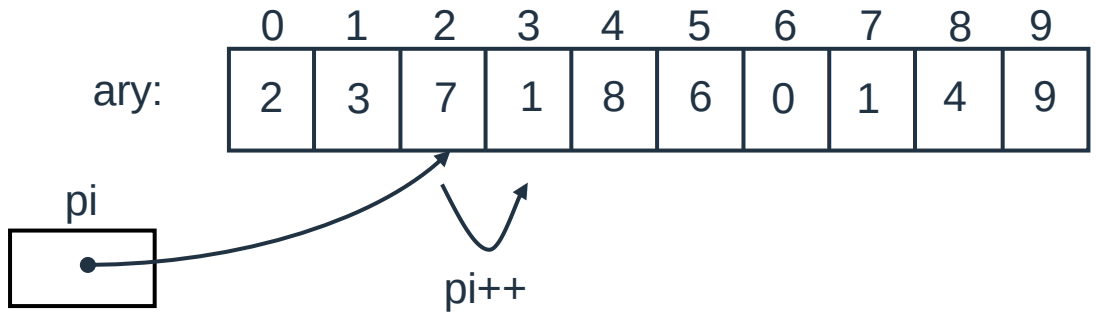
```
int i;
i = *(pi+4);  /* 5e element, let op haakjes */
```

- Maar ook mag:

```
i = pi[4];
```

- Arrays en pointers zijn uitwisselbaar!

Pointers – rekenen



- Array

```
int ary[] = { 2, 3, 7, 1, 8, 6, 0, 1, 4, 9 };
int *pi = ary+2; /* pi wijst naar ary[2] */
```

- Nu verhogen en verlagen:

```
pi++; /* wijst naar volgende element */
pi--; /* wijst naar vorige element */
```

Pointers – array van character

- Array van characters:

```
char str[] = "Hello world!";
```

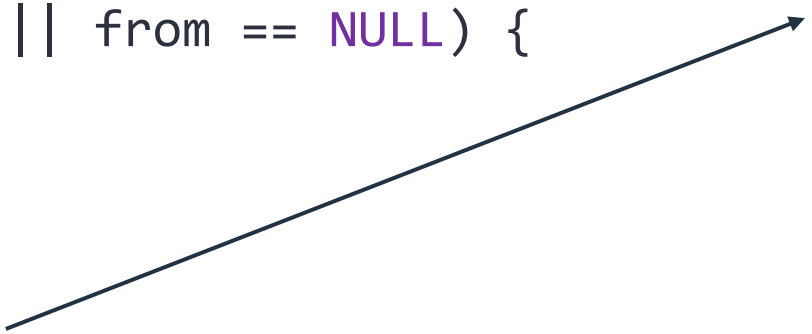
- Lopen langs string:

```
char *pc = str;
while (*pc != '\0') {
    putchar(*pc);  /* druk teken af */
    pc++;
}
```

Pointers – kopieren

- Kopieren van strings:

```
void stringcopy(char *to, char *from) {

    if (to == NULL || from == NULL) {
        return;
    }

    while ((*to++ = *from++) != '\0');
}
```



Hier gebeuren drie dingen:

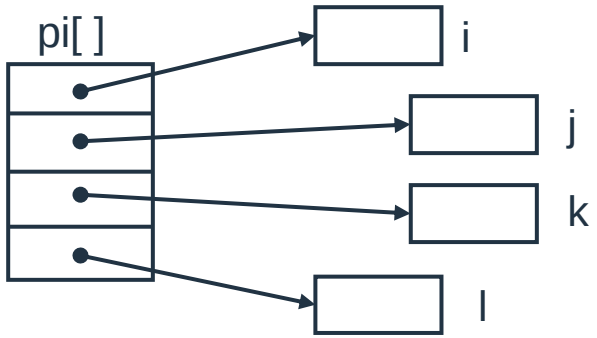
1. De `char` waar `from` naar wijst wordt gekopieerd naar de plek waar `to` heenwijst.
2. `to` en `from` worden beide na gebruik met één verhoogd. Ze wijzen dus naar de opvolgende `char`'s.
3. De toekenning wordt getest op `'\0'` en de lus wordt herhaald zolang de test niet waar is.

Array van pointers

- Array van pointers naar integers:

```
int i, j, k, l;
int *pi[4]; /* array van 4 pointers naar int */
pi[0] = &i; p[1] = &j; p[2] = &k; p[3] = &l;

printf("%d %d %d %d\n", *pi[0], *pi[1], *pi[2], *pi[3]);
```

- In geheugen:

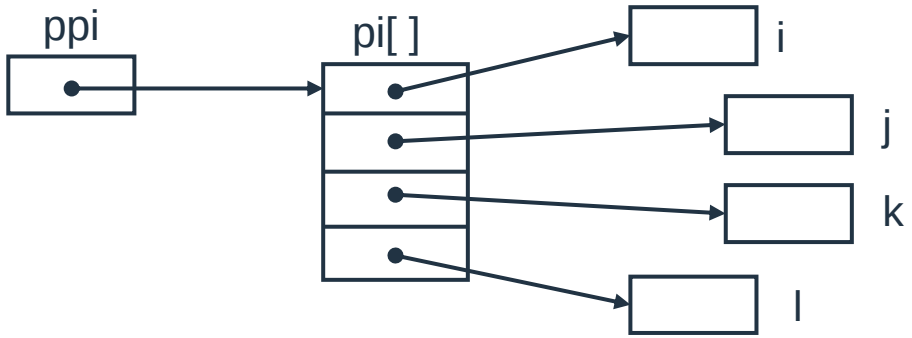


Pointer naar een pointer naar een int

- Pointer naar een pointer naar een integer:

```
int i, j, k, l;
int *pi[4] = { &i, &j, &k, &l };
int **ppi = pi;  /* pointer naar array van pointers */
```

- In geheugen:



DE HAAGSE HOGESCHOOL

Pointer naar een pointer naar een int

- Pointer naar een pointer naar een integer :

```
int i, j, k, l;
int *pi[4] = { &i, &j, &k, &l };
int **ppi = pi;  /* pointer naar array van pointers */
```

- `pi[0]` wijst naar `i`, `pi[1]` wijst naar `j`, `pi[2]` wijst naar `k` en `p[3]` wijst naar `l`.
- `ppi` wijst naar `pi[0]`.
- Via `ppi` kunnen we bij `i`, `j`, `k` en `l` komen.

Pointer naar een pointer naar een int

- Pointer naar een pointer naar een integer :

```
int i, j, k, l;
int *pi[4] = { &i, &j, &k, &l };
int **ppi = pi;  /* pointer naar array van pointers */
```

- `ppi` “weet” niet dat hij naar een array wijst.
- Via `ppi` kunnen we de array afdrukken.

Pointer naar een pointer naar een int

- Afdrukken array met `ppi`:

```c
#include <stdio.h>

int main(void) {

    int i = 2, j = 5, k = 3, l = 8;

    int *pi[4] = { &i, &j, &k, &l };
    int **ppi = pi;

    for (int index = 0; index < 4; index++) {
        printf("%d\n", **ppi++);
    }
}
```

DE HAAGSE HOGESCHOOL

Command line parameters

- De functie `main` wordt normaal gesproken gedeclareerd als:

```
int main(void);
```

- Maar er is een tweede declaratie mogelijk:

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

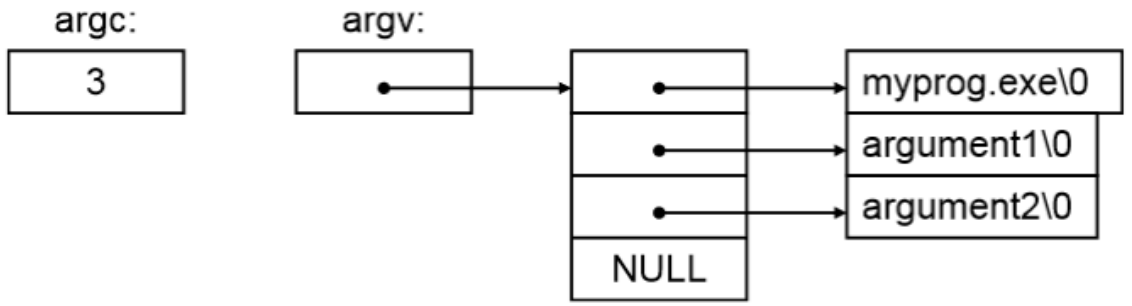
- `argc` is een integer.
- `argv[]` is een array van pointers naar characters.

Command line parameters

- Maar ook mag:

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

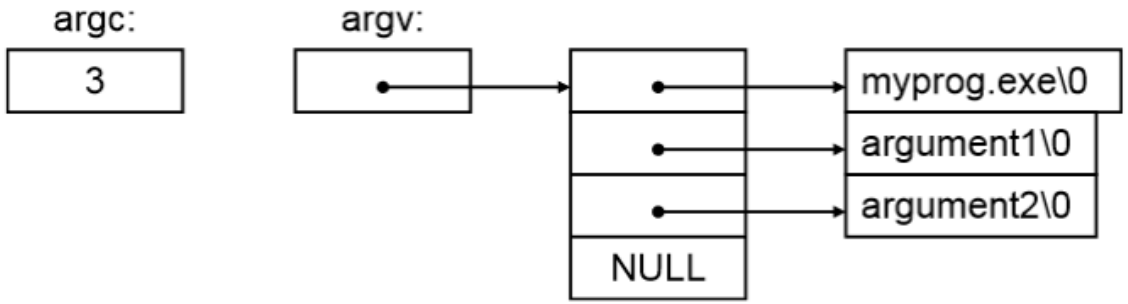
- `argc` is een integer.
- `argv` is een pointer naar een pointer naar een character.



DE HAAGSE HOGESCHOOL

Command line parameters

- Aan een programma kunnen *command line parameters* worden meegegeven.
- De parameters worden als *strings* in het geheugen geplaatst.
- `argc` bevat het aantal parameters en is minstens 1.
- `argv` mag gelezen worden als een pointer naar een array van pointers naar strings.
- De eerste parameter is per definitie de naam van het uitvoerbaar programma.



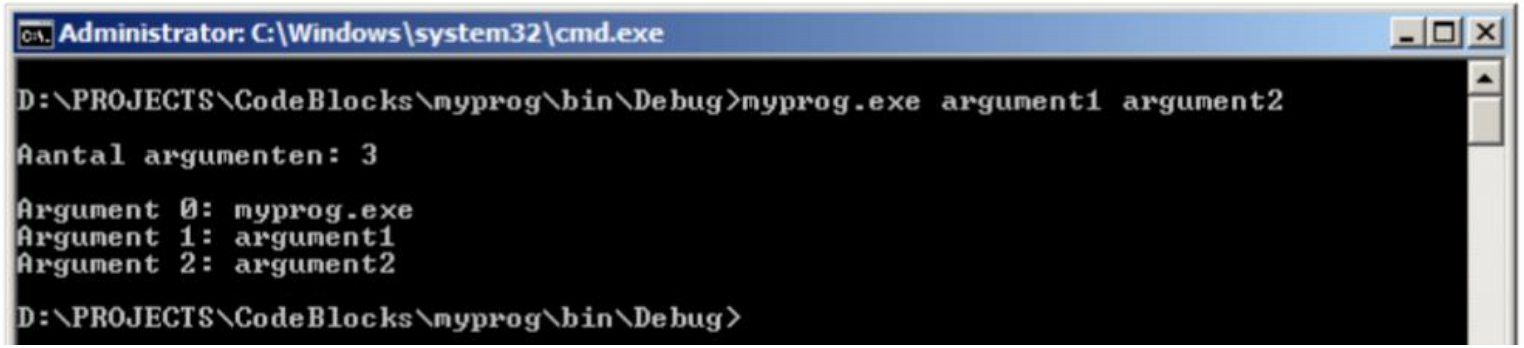
DE HAAGSE HOGESCHOOL

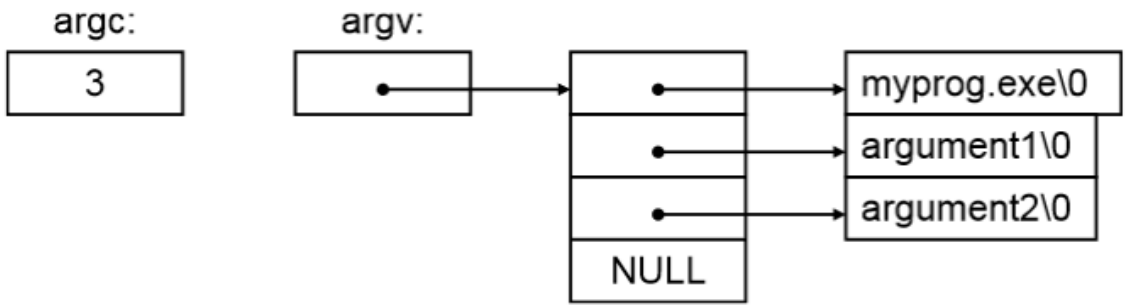
Command line parameters

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

int main(int argc, char *argv[]) {

    int i;

    printf("\nAantal argumenten: %d\n\n", argc);
    for (i = 0; i<argc; i++) {
        printf("Argument %d: %s\n", i, argv[i]);
    }
    return 0;
}
```

Command line parameters

- Voorbeeld van meegegeven parameters:





DE HAAGSE HOGESCHOOL

Pointers naar structs

- Pointer naar structs:

```
struct student_struct {
    int idcode;
    char naam[30];
};

struct student_struct s1 = { 21001234, "Aalberse" };
struct student_struct *ps = &s1;
```

Pointers naar structs

- Afdrukken van de struct:

```
void printstudent(struct student_struct *pstud) {

    printf("Idcode: %d\n", (*pstud).idcode);
    printf("Naam: %s\n", (*pstud).naam);
}
```

- De haakjes bij `*pstud` zijn nodig voor de juiste prioriteit.

Pointers naar structs

- `(*pstud).` … komt zo vaak voor dat er een alternatieve mogelijkheid is.

```
void printstudent(struct student_struct *pstud) {

    printf("Idcode: %d\n", pstud->idcode);
    printf("Naam: %s\n", pstud->naam);
}
```

- Voordeel van pointers is dat niet de hele struct wordt gekopieerd maar alleen een pointer.

Dynamische geheugenallocatie

- In C is het mogelijk om aan het Operating System een *geheugenblok* op te vragen.
- Dit geheugenblok bestaat uit aaneengesloten geheugenplaatsen.

- Opvragen geheugenblok voor 1000 integers:

```
#include <malloc.h>
int *pint = (int *) malloc(1000 * sizeof(int));
```

- `pint` is een pointer naar het eerste geheugenplaats.
- Geheugenblok wordt niet geïnitialiseerd.

Dynamische geheugenallocatie

- Opvragen geheugenblok voor 1000 integers:

```
int *pint = (int *) malloc(1000 * sizeof(int));
```

- Checken of allocatie gelukt is:

```
if (pint == NULL) {
   printf("Allocatie mislukt!\n");
   return -1;
}
```

Dynamische geheugenallocatie

- Opvragen geheugenblok voor 1000 integers:

```
int *pint = (int *) malloc(1000 * sizeof(int));
```

- Vrijgeven van geheugen:

```
free(pint);
```

- Opmerking: `pint` mag niet aangepast worden!

Dynamische geheugenallocatie

- Gebruik van het geheugenblok:

```
int *pint =  (int *) malloc(1000 * sizeof(int));
int *p = pint;

for (int i = 0; i < 1000; i++) {
   *p++ = i;
}

free(pint);
```

Dynamische geheugenallocatie

- Het geheugenblok mag ook als een array gezien worden:

```
int *pint =  (int *) malloc(1000 * sizeof(int));

for (int i = 0; i < 1000; i++) {
    pint[i] = i;  /* of *(pint+i) */
}
```

let’s change