

Pertemuan 13



Pointer

Mata Kuliah : Algoritma & Pemrograman
Dosen : Tessy Badriyah, SKom., MT., PhD.



Pembahasan

- Pembahasan tentang Pointer
- Pointer dan parameter fungsi



Pointer

- Suatu pointer adalah **variable**
- Berisi **alamat memori**
- Menunjuk ke **tipe data** tertentu
- Variabel pointer biasanya diberi nama ***varPtr***



Alamat memori dari suatu variabel

```
char ch = 'A';
```

ch :

0x2000

'A'

Alamat memori
dari variabel *ch*

Nilai / value dari
variabel *ch*



Operator &

- Memberikan alamat memori dari suatu obyek

```
char ch = 'A' ;
```

0x2000

'A'

&ch

Mendapatkan nilai
0x2000

- disebut juga “alamat operator”



Contoh:

```
char ch;
```

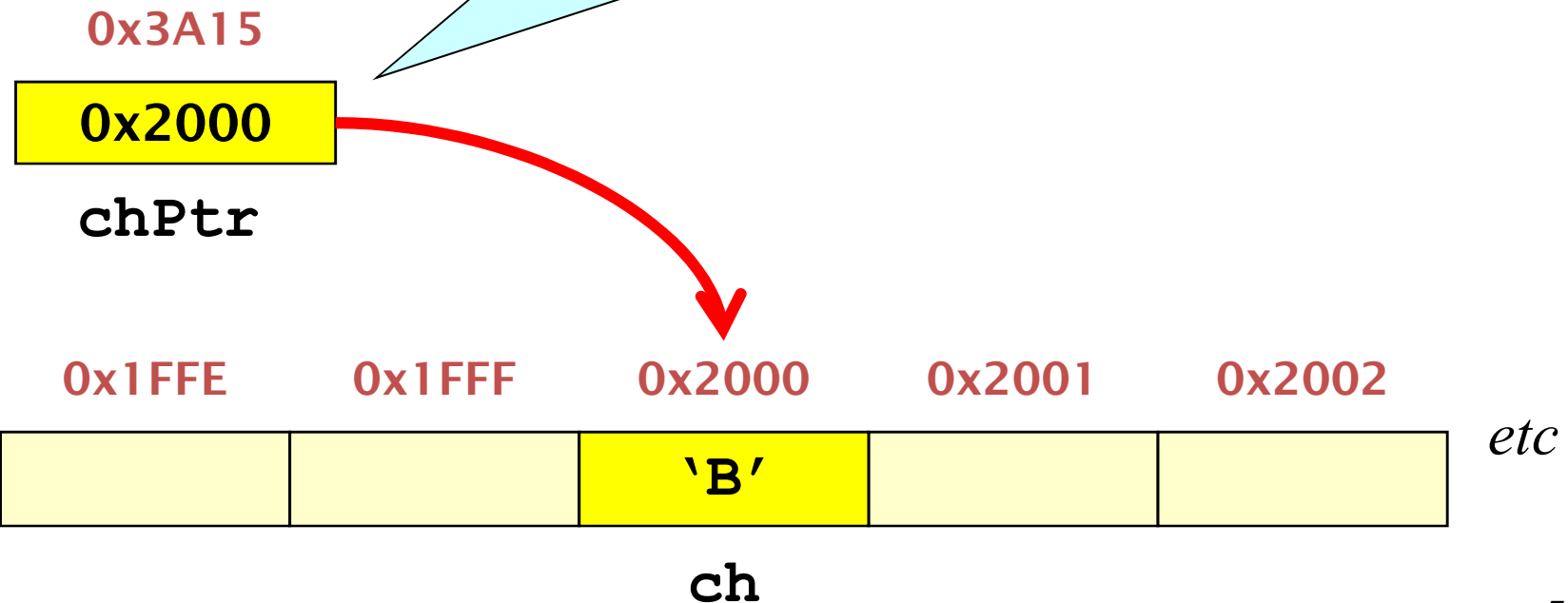
```
printf("%p", &ch);
```

***“conversion specifier” untuk
mencetak alamat memori***



Pointers

Variabel yang menyimpan
alamat memori dari
variabel lain



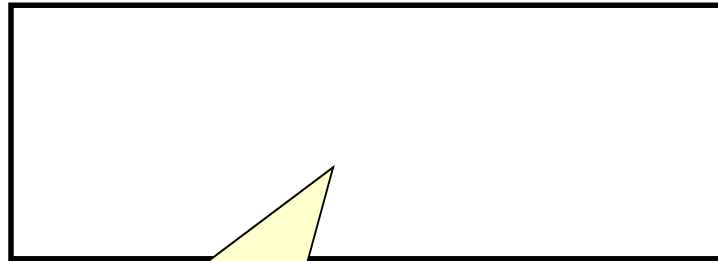


Contoh:

```
char* cPtr;
```

cPtr:

0x2004



Dapat menyimpan **alamat** dari
variable bertipe **char**

- Kita katakan *cPtr* adalah ***pointer*** ke tipe data char



Pointer dan Operator &

Contoh:

```
char c = 'A' ;
```

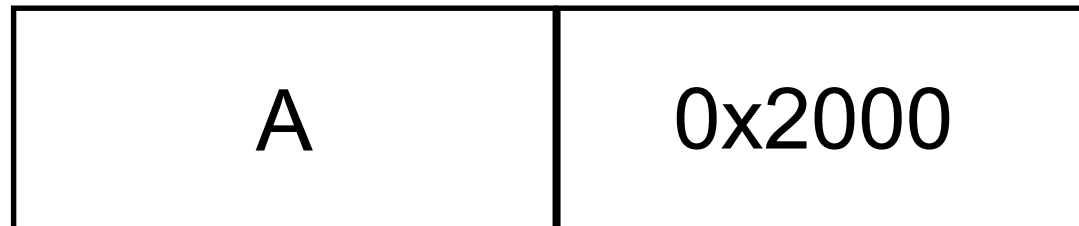
```
char *cPtr;
```

```
cPtr = &c;
```

*Menandai (assign)
alamat dari **c** ke
cPtr*

c:

cPtr:



0x2000

0x2004



Catatan tentang Pointer

- Kita dapat memiliki pointer untuk sembarang tipe data

Contoh:

```
int*    numPtr;  
float*  xPtr;
```

- Tanda * dapat diletakkan di sembarang tempat antara tipe data dan variabel

Contoh:

```
int      *numPtr;  
float *  xPtr;
```



Catatan tentang Pointer

- Alamat dari variable dapat di-assign ke pointer yang “**compatible**” dengan menggunakan operator **&**

Contoh:

```
int    aNumber;  
int    *numPtr;  
  
numPtr = &aNumber;
```

- Alamat yang disimpan dalam pointer dapat dicetak dengan menggunakan conversion specifier **%p**

Contoh:

```
printf("%p", numPtr);
```



Operator *

- Memungkinkan pointer untuk mengakses variable yang ditunjuk
- Disebut juga sebagai “dereferencing operator”
- Jangan disamakan / disalah artikan dengan tanda * pada deklarasi pointer



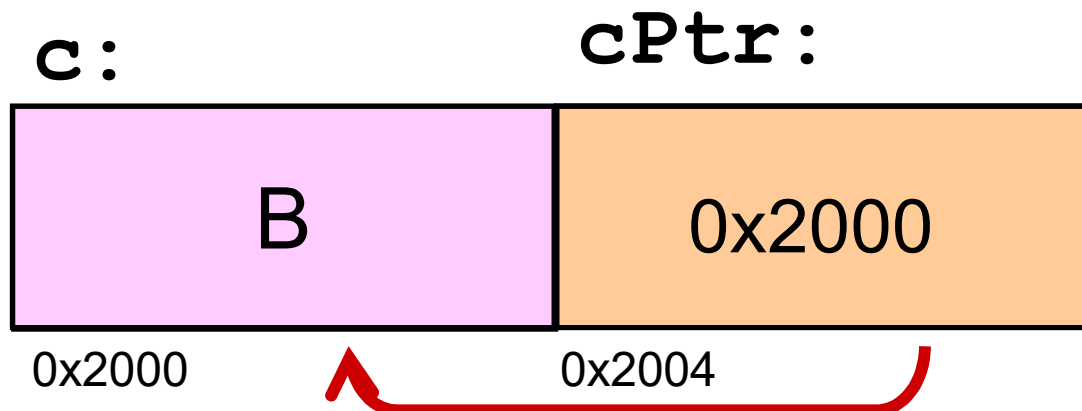
Pointer dan Operator *

Contoh:

```
char c = 'A';  
char *cPtr = NULL;
```

```
cPtr = &c;  
*cPtr = 'B';
```

*Perubahan dari nilai
variable yang ditunjuk
oleh pointer **cPtr***




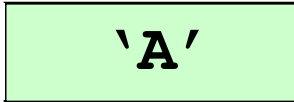


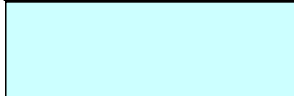
Cara mudah menggunakan Pointers

- **Step 1:** Deklarasikan variable yang akan ditunjuk oleh pointer

```
int num;  
char ch = 'A';  
float x;
```

num: 

ch: 

x: 



Cara mudah menggunakan Pointers

- **Step 2:** Deklarasikan variable pointer

```
int num;  
char ch = 'A';  
float x;
```

```
int* numPtr = NULL;  
char *chPtr = NULL;  
float *xPtr = NULL;
```

numPtr:

NULL

chPtr:

NULL

xPtr:

NULL

num:

--

ch:

'A'

x:

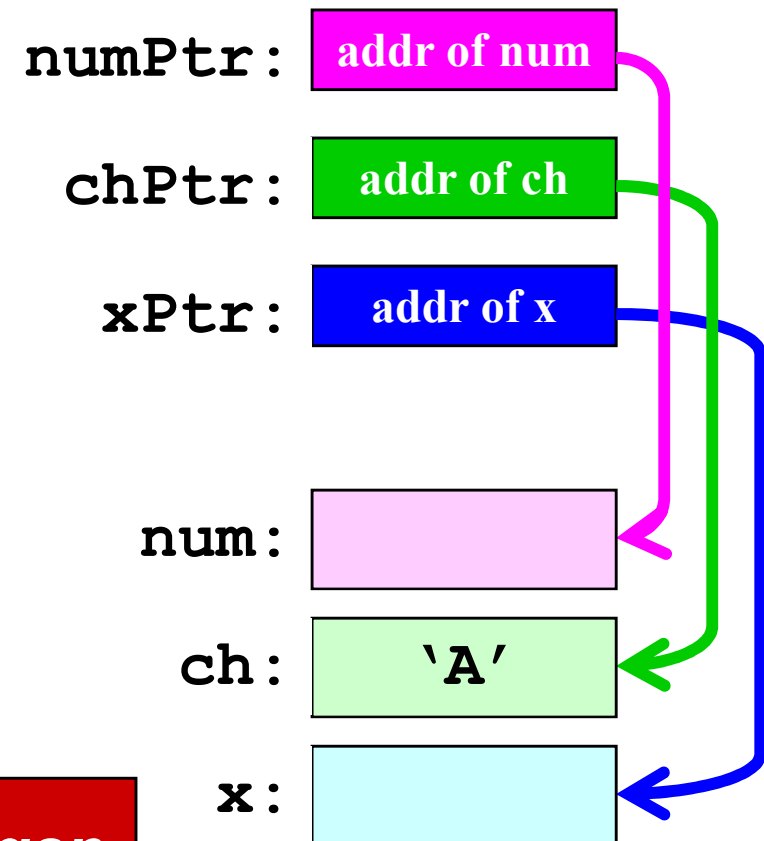
--

Cara mudah menggunakan Pointers

- **Step 3:** Assign alamat dari variable ke pointer

```
int    num;  
char   ch = 'A';  
float  x;  
  
int*   numPtr = NULL;  
char*  chPtr  = NULL;  
float* xPtr   = NULL;
```

```
numPtr = &num;  
chPtr  = &ch;  
xPtr   = &x;
```



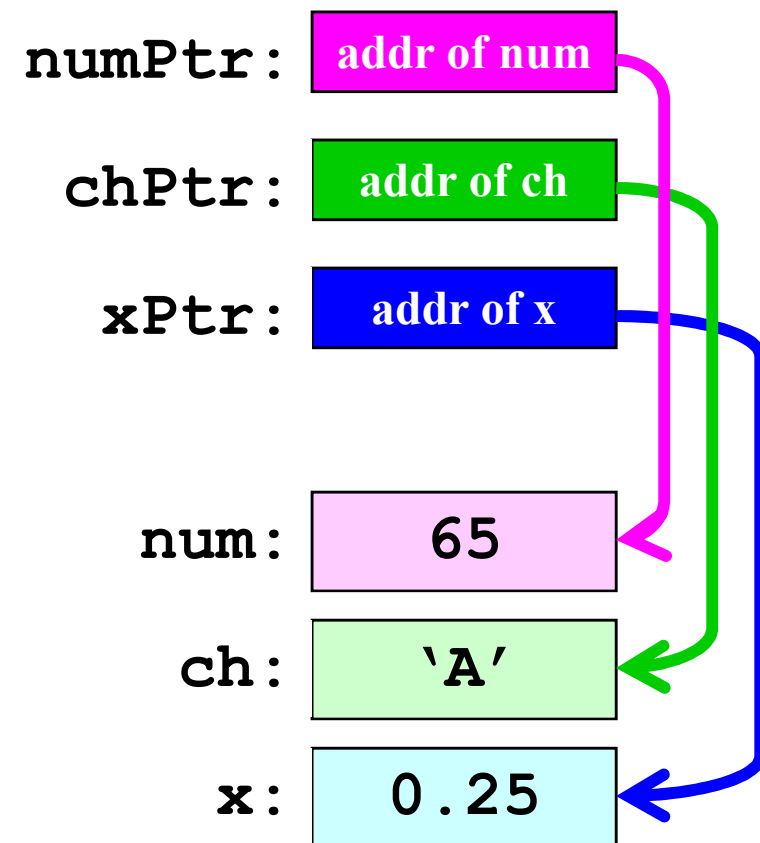
Tipe data pointer harus sama dengan tipe data variable yang ditunjuk



Cara mudah menggunakan Pointers

- **Step 4: De-reference pointer**

```
int    num;  
char   ch = 'A';  
float  x;  
  
int*   numPtr = NULL;  
char*  chPtr  = NULL;  
float* xPtr   = NULL;  
  
numPtr = &num;  
chPtr  = &ch;  
xPtr   = &x;  
  
*xPtr = 0.25;  
*numPtr = *chPtr;
```

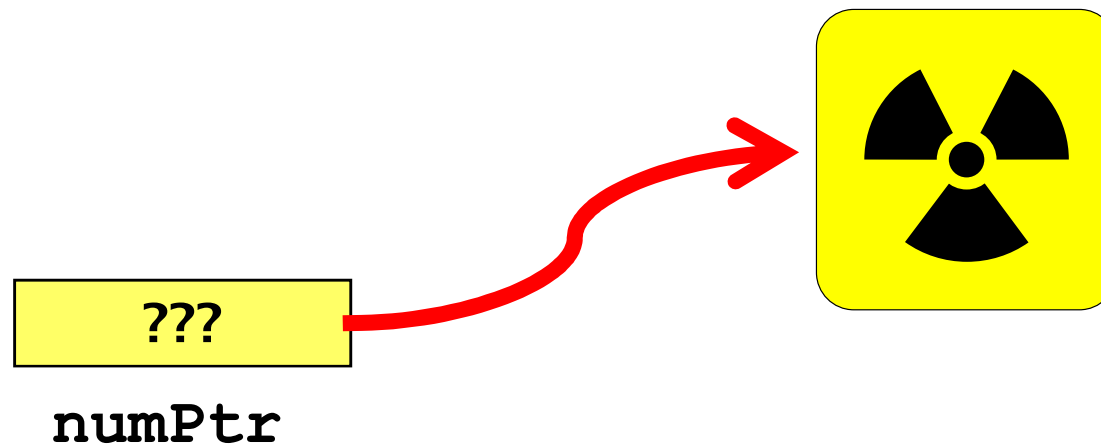




Catatan tentang Pointer

```
int *numPtr;
```

***Hati-hati dengan
pointer yang tidak
diinisialisasi***





Catatan tentang Pointer

- Saat mendeklarasikan suatu pointer, lebih baik menginisialisasinya dengan nilai **NULL** (konstanta khusus dari pointer)

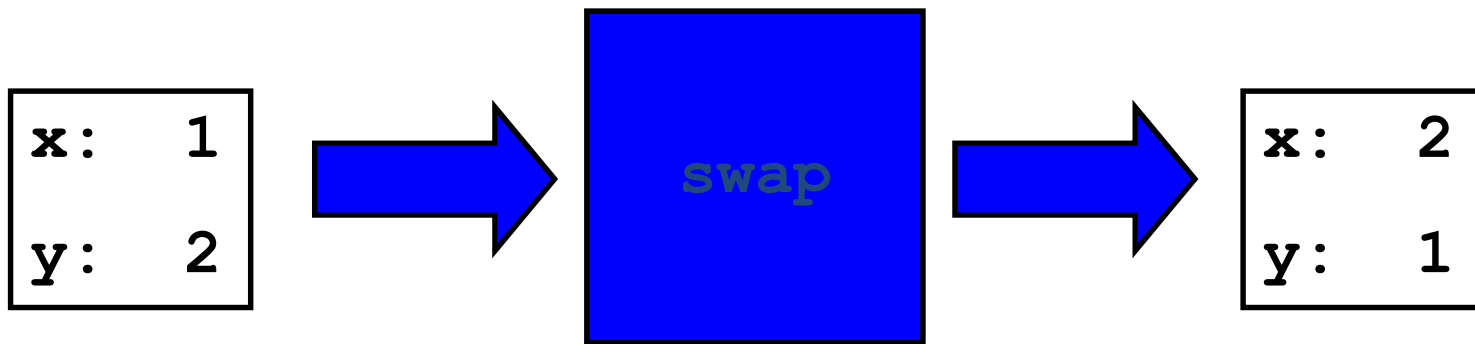
```
int *numPtr = NULL;
```

NULL

numPtr

Pointers dan Parameter Fungsi

- **Contoh:** Fungsi untuk menukar nilai dari dua variabel





Bad swap

```
#include <stdio.h>

void swap1(int a, int b)
{
    int tmp;

    tmp = a;
    a = b;
    b = tmp;
    return;
}

int main()
{
    int x = 1, y = 2;

    swap1(x, y);
    printf("%d %d\n", x, y);
    return 0;
}
```



Bad swap

```
#include <stdio.h>

void swap1(int a, int b)
{
    int tmp;

    tmp = a;
    a = b;
    b = tmp;
    return;
}

int main()
{
    int x = 1, y = 2;

    swap1(x, y);
    printf("%d %d\n", x, y);
    return 0;
}
```

x:

1

y:

2



Bad swap

```
#include <stdio.h>

void swap1(int a, int b)
{
    int tmp;

    tmp = a;
    a = b;
    b = tmp;
    return;
}

int main()
{
    int x = 1, y = 2;

    swap1(x, y);
    printf("%d %d\n", x, y);
    return 0;
}
```

tmp:

a:

b:

x:

y:



Bad swap

```
#include <stdio.h>

void swap1(int a, int b)
{
    int tmp;

    tmp = a;
    a = b;
    b = tmp;
    return;
}

int main()
{
    int x = 1, y = 2;

    swap1(x, y);
    printf("%d %d\n", x, y);
    return 0;
}
```

tmp:

1

a:

1

b:

2

x:

1

y:

2



Bad swap

```
#include <stdio.h>

void swap1(int a, int b)
{
    int tmp;

    tmp = a;
    a = b;
    b = tmp;
    return;
}

int main()
{
    int x = 1, y = 2;

    swap1(x, y);
    printf("%d %d\n", x, y);
    return 0;
}
```

tmp:

1

a:

2

b:

2

x:

1

y:

2



Bad swap

```
#include <stdio.h>

void swap1(int a, int b)
{
    int tmp;

    tmp = a;
    a = b;
    b = tmp;
    return;
}

int main()
{
    int x = 1, y = 2;

    swap1(x, y);
    printf("%d %d\n", x, y);
    return 0;
}
```

tmp:

1

a:

2

b:

1

x:

1

y:

2



Bad swap

```
#include <stdio.h>

void swap1(int a, int b)
{
    int tmp;

    tmp = a;
    a = b;
    b = tmp;
    return;
}

int main()
{
    int x = 1, y = 2;

    swap1(x, y);
    printf("%d %d\n", x, y);
    return 0;
}
```

tmp:

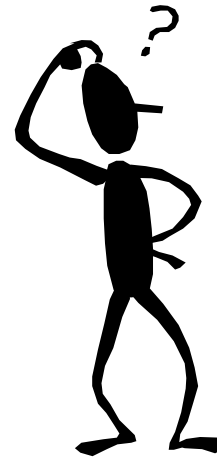
1

a:

2

b:

1



x:

1

y:

2



Good swap

```
#include <stdio.h>

void swap2(int* a, int* b)
{
    int tmp;

    tmp = *a;
    *a = *b;
    *b = tmp;
    return;
}

int main()
{
    int x = 1, y = 2;

    swap2(&x, &y);
    printf("%d %d\n", x, y);
    return 0;
}
```



Good swap

```
#include <stdio.h>

void swap2(int* a, int* b)
{
    int tmp;

    tmp = *a;
    *a = *b;
    *b = tmp;
    return;
}

int main()
{
    int x = 1, y = 2;

    swap2(&x, &y);
    printf("%d %d\n", x, y);
    return 0;
}
```

x:

1

y:

2

Good swap

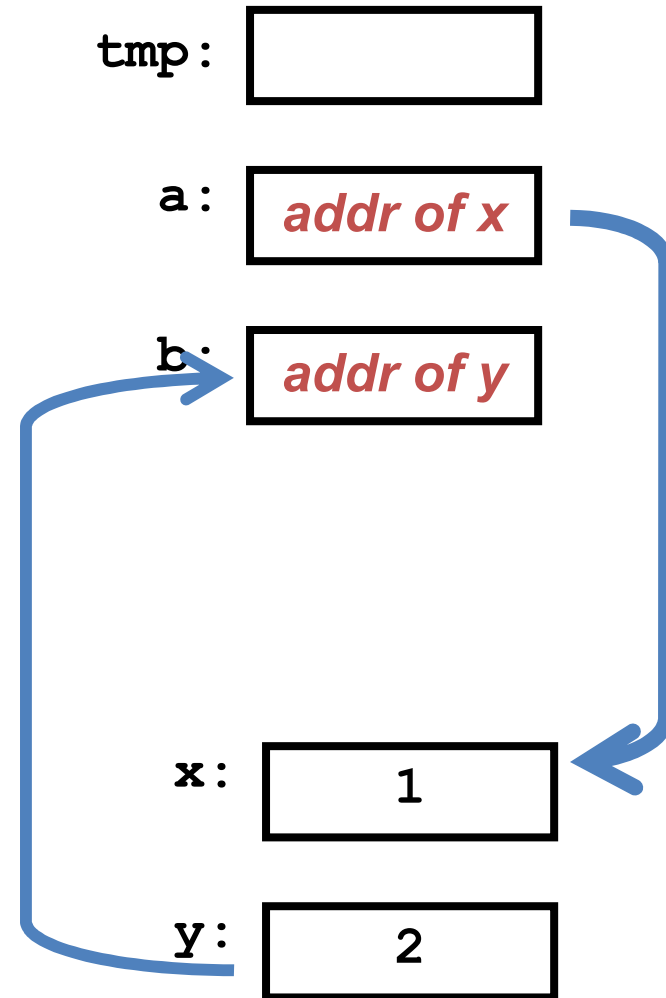
```
#include <stdio.h>

void swap2(int* a, int* b)
{
    int tmp;

    tmp = *a;
    *a = *b;
    *b = tmp;
    return;
}

int main()
{
    int x = 1, y = 2;

    swap2(&x, &y);
    printf("%d %d\n", x, y);
    return 0;
}
```





Good swap

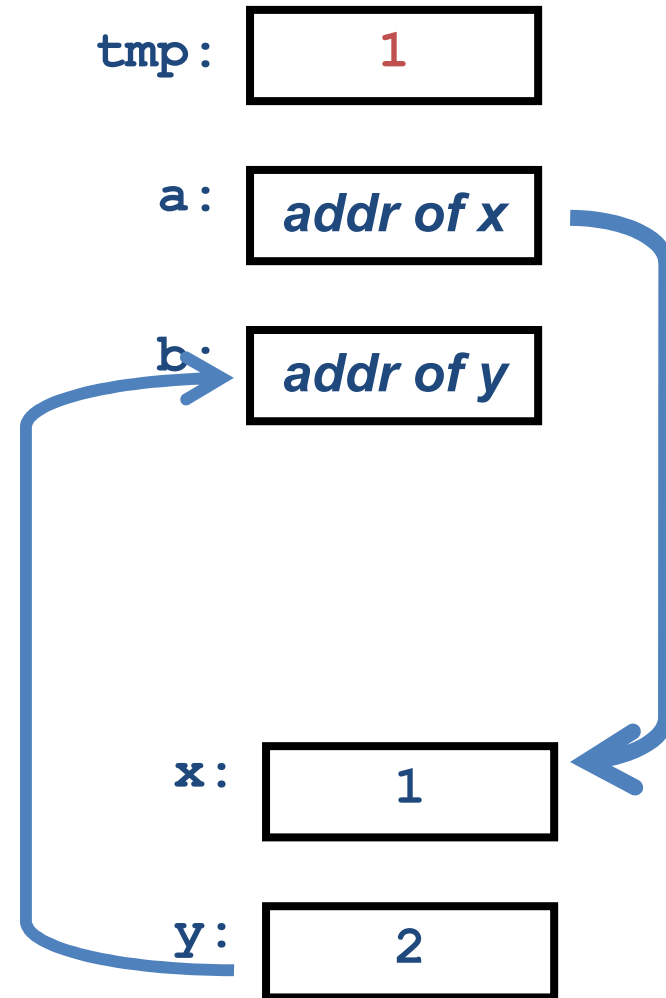
```
#include <stdio.h>

void swap2(int* a, int* b)
{
    int tmp;

    tmp = *a;
    *a = *b;
    *b = tmp;
    return;
}

int main()
{
    int x = 1, y = 2;

    swap2(&x, &y);
    printf("%d %d\n", x, y);
    return 0;
}
```





Good swap

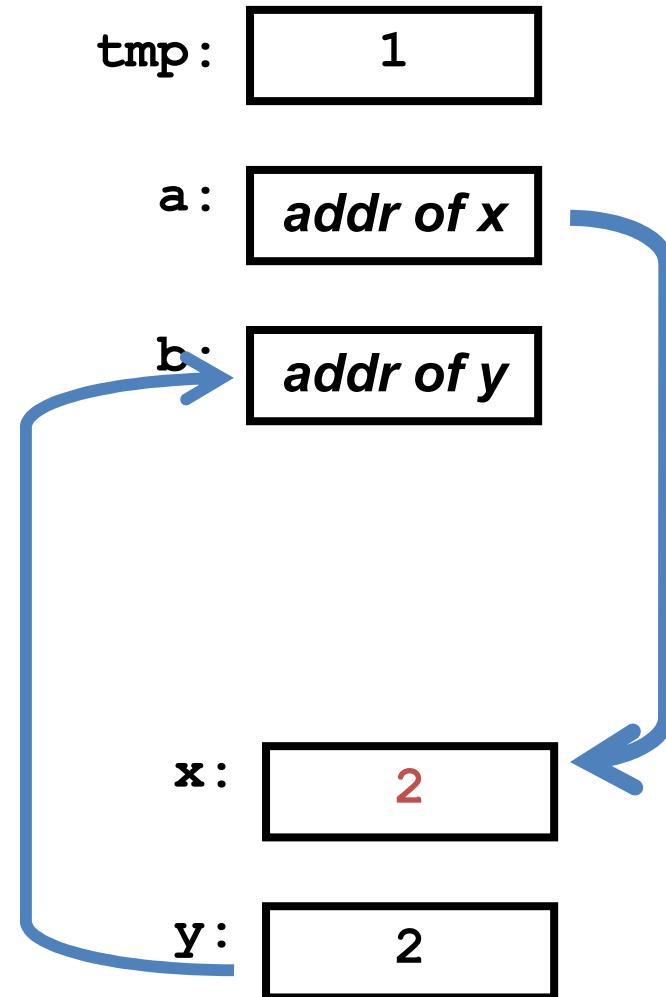
```
#include <stdio.h>

void swap2(int* a, int* b)
{
    int tmp;

    tmp = *a;
    *a = *b;
    *b = tmp;
    return;
}

int main()
{
    int x = 1, y = 2;

    swap2(&x, &y);
    printf("%d %d\n", x, y);
    return 0;
}
```





Good swap

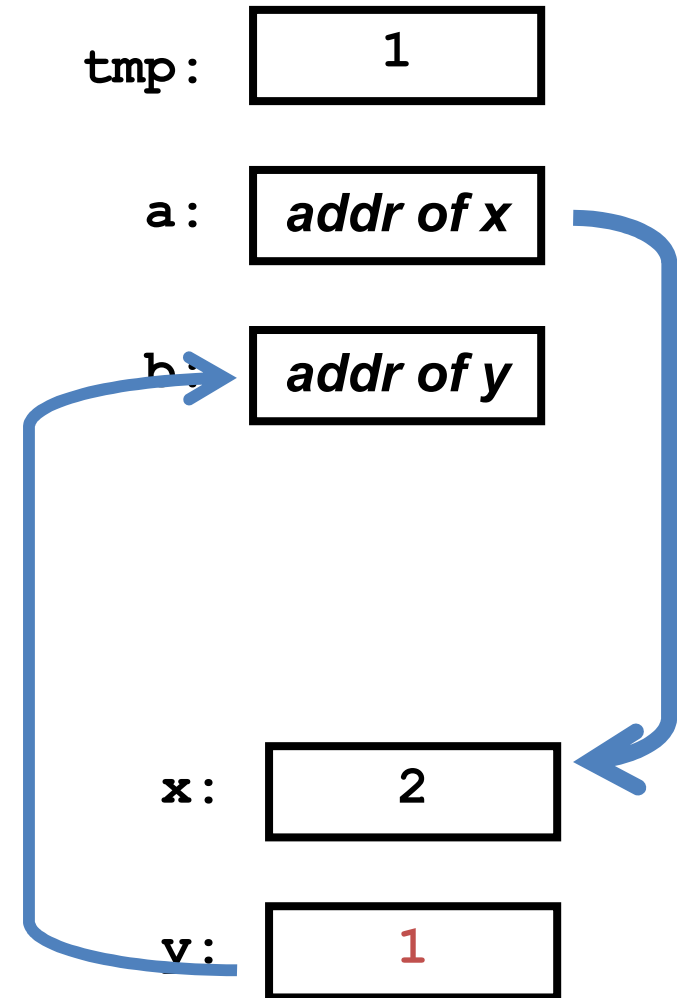
```
#include <stdio.h>

void swap2(int* a, int* b)
{
    int tmp;

    tmp = *a;
    *a = *b;
    *b = tmp;
    return;
}

int main()
{
    int x = 1, y = 2;

    swap2(&x, &y);
    printf("%d %d\n", x, y);
    return 0;
}
```





Good swap

```
#include <stdio.h>

void swap2(int* a, int* b)
{
    int tmp;

    tmp = *a;
    *a = *b;
    *b = tmp;
    return;
}

int main()
{
    int x = 1, y = 2;

    swap2(&x, &y);
    printf("%d %d\n", x, y);
    return 0;
}
```



x:

2

y:

1



Pointer dan Argumen Fungsi

- Merubah nilai dari parameter aktual

- **scanf**

```
char ch;  
int  numx;  
float numy;  
scanf("%c %d %f", &ch, &numx, &numy);
```





Yang akan dipelajari berikutnya

- Review dasar pointer
- Lebih lanjut tentang bagaimana men- dereference sebuah pointer
- Lebih lanjut tentang melewati parameter passing by reference pada fungsi
- Lebih lanjut tentang contoh-contoh pointer

- Sebuah pointer *menunjuk* ke variable lain
- Sebuah pointer berisi *alamat* dari variable lain
- Operator `*` digunakan untuk mendeklarasikan pointer
 - contoh `int* xPtr;`
- Operator `&` memberikan alamat dari variabel
- Operator `*` men-*dereferences* sebuah pointer – dan memberi nilai dari sesuatu yang ditunjuk oleh pointer.



Mengulang kembali

- Pointer dideklarasikan dalam tipe data yang spesifik.
- Sembarang tipe data dapat memiliki pointer
- Seperti variable lainnya, tipe dari pointer bersifat tetap
 - Jadi jika suatu variable dideklarasikan sebagai **char*** maka akan *selalu* menunjukkan ke variable bertipe data **char**



Lebih lanjut tentang Dereferencing

- Sebuah pointer *menunjuk* ke variable lain
- Kita perlu pointer untuk menemukan nilai dari variable yang ditunjuk.
- Kita bisa melakukan dengan cara *dereferencing* pointer dengan menggunakan operator *
- Tapi apa yang sebenarnya terjadi pada saat kita melakukan dereference?



Algoritma untuk Dereferencing

- Untuk me- *dereference* sebuah pointer, gunakan `*xPtr`, yang artinya:

1. Ke `xPtr`
2. Ambil nilai yang didapat disana, jadikan sebagai alamat
3. Ke alamat tersebut
4. Return nilai (value) yang ditemukan disana.



Contoh Pointer

```
char ch;
```

```
ch = 'A';
```

ch :

0x2000

'A'

```
char* chPtr=NULL;
```

```
chPtr=&ch;
```

chPtr :

0x2004

0x2000



Contoh Pointer

```
char ch='A' ;
```

```
char init='B' ;
```

```
char* cPtr1=NULL;
```

```
char* cPtr2=NULL;
```

```
cPtr1=&ch;
```

```
cPtr2=&init;
```

```
*cPtr1 berisi??
```

```
*cPtr2 berisi??
```

```
cPtr1 berisi??
```

```
cPtr2 berisi??
```

0x2000

'A'

ch

0x2001

'B'

init

0x2002

0x2000

cPtr1

0x2003

0x2001

cPtr2



Contoh Pointer

```
char ch='A' ;
```

```
char init='B' ;
```

```
char* cPtr1=NULL;
```

```
char* cPtr2=NULL;
```

```
cPtr1=&ch;
```

```
cPtr2=&init;
```

```
*cPtr1 berisi 'A'
```

```
*cPtr2 berisi??
```

```
cPtr1 berisi??
```

```
cPtr2 berisi??
```

0x2000

'A'

ch

0x2001

'B'

init

0x2002

0x2000

cPtr1

0x2003

0x2001

cPtr2



Contoh Pointer

```
char ch='A';
```

```
char init='B';
```

```
char* cPtr1=NULL;
```

```
char* cPtr2=NULL;
```

```
cPtr1=&ch;
```

```
cPtr2=&init;
```

```
*cPtr1 berisi 'A'
```

```
*cPtr2 berisi 'B'
```

```
cPtr1 berisi ??
```

```
cPtr2 berisi ??
```

0x2000

'A'

ch

0x2001

'B'

init

0x2002

0x2000

cPtr1

0x2003

0x2001

cPtr2



Contoh Pointer

```
char ch='A' ;
```

```
char init='B' ;
```

```
char* cPtr1=NULL;
```

```
char* cPtr2=NULL; 0x2001
```

```
cPtr1=&ch;
```

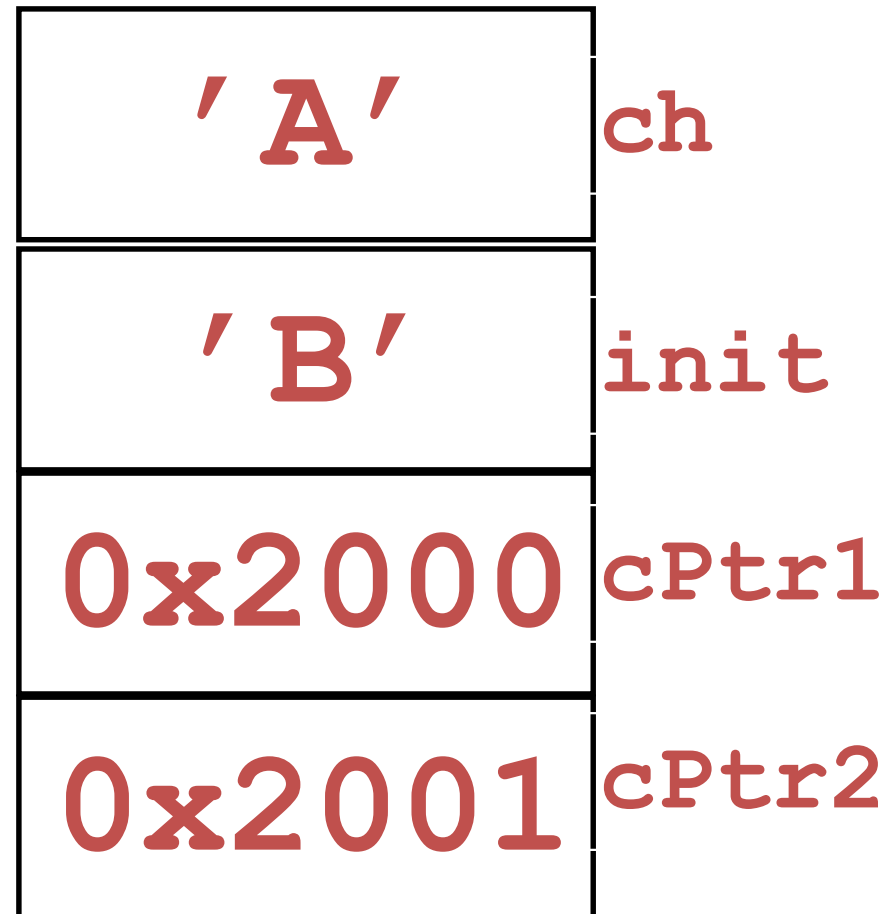
```
cPtr2=&init; 0x2002
```

```
*cPtr1 berisi 'A'
```

```
*cPtr2 berisi 'B' 0x2003
```

```
cPtr1 berisi 0x2000
```

```
cPtr2 berisi??
```





Contoh Pointer

```
char ch='A' ;
```

```
char init='B' ;
```

```
char* cPtr1=NULL;
```

```
char* cPtr2=NULL; 0x2001
```

```
cPtr1=&ch;
```

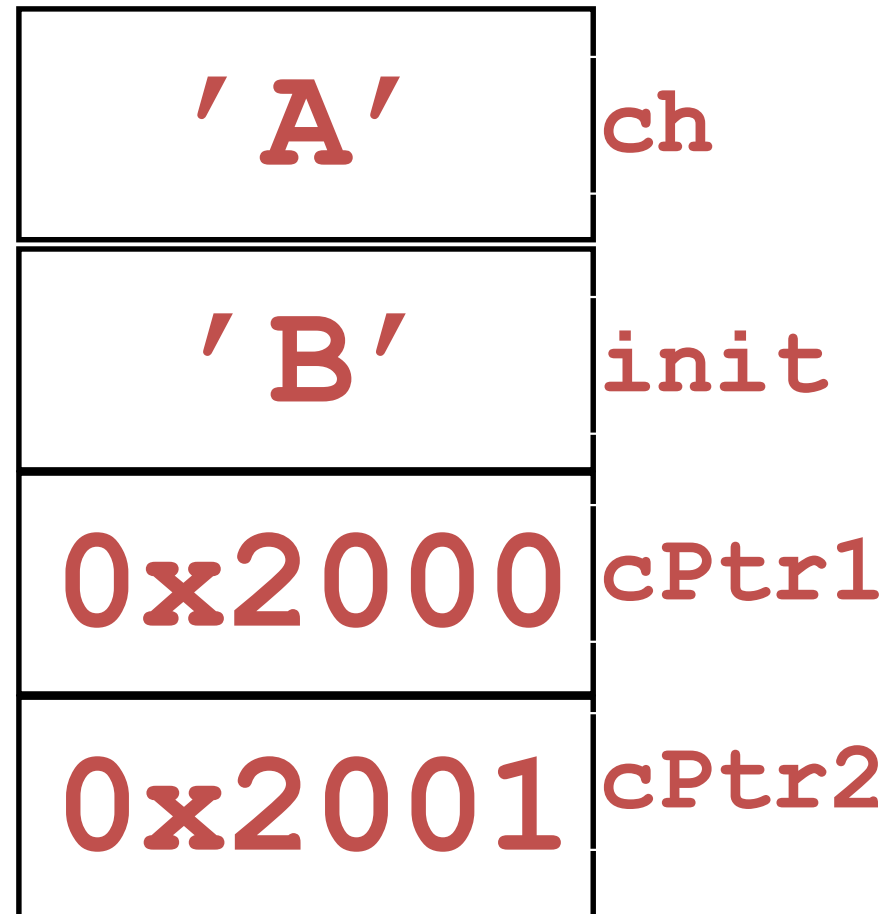
```
cPtr2=&init; 0x2002
```

```
*cPtr1 berisi 'A'
```

```
*cPtr2 berisi 'B' 0x2003
```

```
cPtr1 berisi 0x2000
```

```
cPtr2 berisi 0x2001
```





Contoh Pointer

```
char ch='A' ;
```

```
char init='B' ;
```

```
char* cPtr1=NULL;
```

```
char* cPtr2=NULL;
```

```
cPtr1=&ch;
```

```
cPtr2=&init;
```

```
...
```

```
cPtr1=&init;
```

```
*cPtr1 berisi ??
```

0x2000

'A'

ch

0x2001

'B'

init

0x2002

0x2000

cPtr1

0x2003

0x2001

cPtr2



Di-assign ke pointer

- Pointer sama seperti variable biasa, yang bisa memiliki type “alamat dari <beberapa tipe>”
- Pointer dapat di-assign ke alamat dari sembarang variable
- Nilai dari suatu pointer dapat berubah, sama seperti nilai dari variable yang lain.
- Nilai dari suatu pointer dapat dimanipulasi, sama seperti nilai dari variable yang lain



Contoh Pointer

```
char ch='A' ;
```

0x2000

'A'

ch

```
char init='B' ;
```

0x2001

'B'

init

```
char* cPtr1=NULL;
```

```
char* cPtr2=NULL;
```

```
cPtr1=&ch;
```

0x2002

0x2000

cPtr1

```
cPtr2=&init;
```

```
*cPtr2='.' ;
```

0x2003

0x2001

cPtr2



Contoh Pointer

```
char ch='A' ;
```

```
char init='B' ; 0x2000
```

```
char* cPtr1=NULL; 0x2001
```

```
char* cPtr2=NULL;
```

```
cPtr1=&ch; 0x2002
```

```
cPtr2=&init;
```

```
*cPtr2='.' ; 0x2003
```

```
cPtr2 berisi ??
```

'A'	ch
'.'	init
0x2000	cPtr1
0x2001	cPtr2



Contoh Pointer

```
char ch='A' ; 0x2000
```

```
char init='B' ;
```

```
char* cPtr1=NULL; 0x2001
```

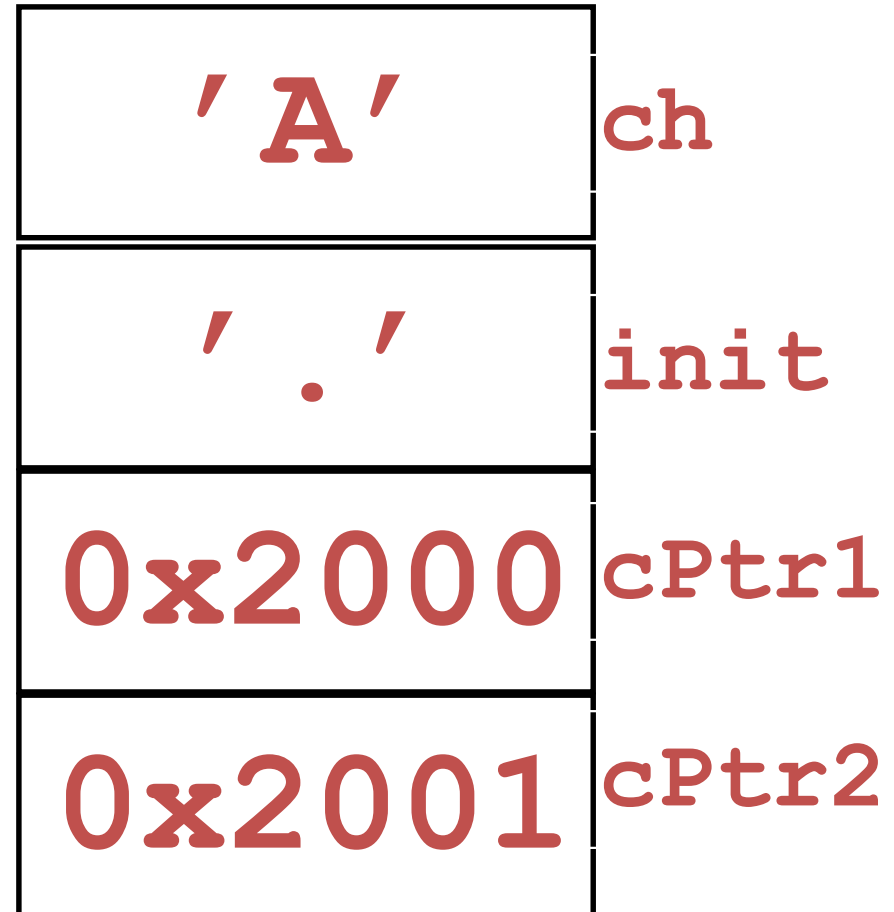
```
char* cPtr2=NULL; 0x2002
```

```
cPtr1=&ch;
```

```
cPtr2=&init;
```

```
*cPtr2='.' ; 0x2003
```

cPtr2 berisi 0x2001





Pointers sebagai Parameters (1)

- Biasanya parameter dilewatkan sebagai kopi dari isinya
- Ini disebut dengan *passing by value*
- Akan tetapi jika kita melewatkan alamat dari variable, maka kita meminta fungsi untuk menemukan nilai actual dari variable, tidak hanya mengkopi isinya saja.




Pointers sebagai Parameters (2)

- Melewatkan parameter sebagai address (alamat) nya berarti kita bisa mendapatkan alamatnya dan merubah isinya jika diperlukan.
- Cara pemanggilan parameter seperti ini disebut dengan *passing by reference*
- Dengan pass by reference, kita bisa merubah nilai dari variable asal karena kita sudah mendapatkan alamatnya.
- Sedangkan jika pass by value, maka yang kita dapatkan hanya isi atau nilai dari variabelnya – dan akan bisa merubah apa-apa (tidak akan bisa merubah nilainya).



Keuntungan dari penggunaan passing by reference

- Efisien
 - Kita tidak perlu membuang waktu dengan membuat tambahan kopi variable setiap kali memanggil fungsi.
- Cara lain untuk mengembalikan (return) sebuah nilai dari fungsi
 - Dan kita bisa mengembalikan lebih dari satu nilai!



Kerugian dari penggunaan passing by reference

- Sulit melacak perubahan yang terjadi pada nilai dari suatu variabel
 - Karena perubahan bisa terjadi dimanapun
- Fungsi jadi bisa mengembalikan lebih dari satu nilai
 - Biasanya fungsi identik pengembalian hanya satu nilai saja, tapi ini tidak maka perubahan lebih dari satu nilai jadi merubah kebiasaan tadi



Contoh tambahan penggunaan pointer

```
int i=0;

int* myPtr=NULL;

int x=3;

myPtr=&x;  /* set myPtr to point to x */

*myPtr=34; /* set x to be 34, using myPtr */

myPtr=&i;  /* set myPtr to point to i */

printf("%d",*myPtr); /* print i using myPtr */

printf("%p",myPtr); /* print the address of i */
```




Contoh tambahan penggunaan pointer

```
float x=5.4,y=78.25;
```

```
float* xPtr=NULL;
```

```
float* yPtr=NULL;
```

```
xPtr=&x;    /* set xPtr to point to x */
```

```
yPtr=&y;    /* set yPtr to point to y */
```

```
*xPtr=*yPtr; /* put the value of y in x using  
pointers */
```

```
*yPtr=45.0   /* put 45.0 in y using yPtr */
```



Yang sudah dipelajari

- Pembahasan tentang Pointer
- Pointer dan parameter fungsi
- Men-dereference sebuah pointer
- Melewatkan parameter passing by reference pada fungsi
- Contoh-contoh penggunaan pointer lebih lanjut

- Robertson, Lesley Anne. (1992). *Students' guide to program design*. Oxford : Newnes
- Santner, Williams, and Notz (2003), *Design and Analysis of Computer Experiments*, Springer.
- Deitel & Deitel, *C How to Program*, Prentice Hall 1994 (2nd edition)
- Brookshear, J.G., *Computer Science: An Overview*, Benjamin-Cummings 2000 (6th edition)
- Kernighan & Ritchie, *The C Programming Language*, Prentice Hall