C pretprocesor

- Uključivanje datoteka

```
#include "imedatoteke"
```

ili

```
#include <imedatoteke>
```

- Lokacija datoteke *imedatoteke*
- Obezbeđuje se bolje povezivanje deklaracija u velikim programima, pošto se na ovaj način garantuje ista deklaracija i definicija u svim izvornim datotekama

C pretprocesor

- Zamena makroa

```
#define   naziv   tekstzamene
#define forever for(;;) /* infinite loop */
```

- Makroi sa argumentima

```
#define max(A, B) ((A) > (B) ? (A) : (B)) /* maksimum od dva broja */
x = max(p+q, r+s);
```

zamenjuje se sa

```
x = ((p+q) > (r+s) ? (p+q) : (r+s));

max(i++, j++) /* nece dati ocekivani rezultat */
#define square(x) x * x /* nece dati ocekivani rezultat */
```

C pretprocesor

- Definicije imena direktive se mogu poništiti

```
#undef max

int max(int,int,int) {...}
```

- Formalni parametri se ne zamenjuju unutar stringa pod navodnicima, što se prevazilazi korišćenjem znaka #

```
#define dprintf(expr)  printf(#expr " = %g\n",expr)
```

Poziv

```
dprint(x/y);
```

se razvija u

```
printf("x/y" " = %g\n",x/y); tj. printf("x/y = %g\n",x/y);
```

C pretprocesor

- Uslovno uključivanje datoteka

  #if, **#else**, **#elif**, **#endif**, **#ifdef**, **#ifndef**

- **Primer**: Uključivanje datoteke hdr.h samo jednom

```
#if !defined(HDR)
   #define HDR
   /* sadrzaj datotek hdr.h ide ovde */
#endif
```

- **Primer**: Uključivanje različitih datoteka u zavisnosti od sistema

```
#if SYSTEM == SYSV
   #define HDR "sysv.h"
#elif SYSTEM == BSD
   #define HDR "bsd.h"
#elif SYSTEM == MSDOS
   #define HDR "msdos.h"
#else
   #define HDR "default.h"
#endif
#include HDR
```

- **`#ifdef` i `#ifndef`**

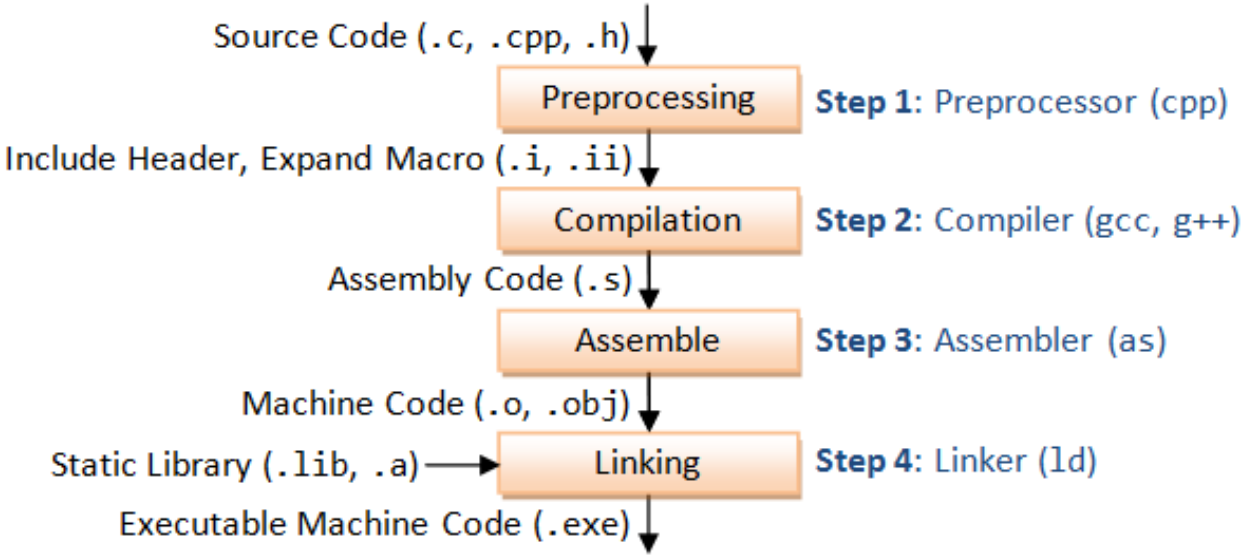
```
#ifndef HDR
   #define HDR
   /* sadrzaj datotek hdr.h ide ovde */
#endif
```

Od izvorne do izvršne datoteka

hello.c

```
#include <stdio.h>

int main() {
    printf("Hello, world!\n");
    return 0;
}
```

- Direktno pravljenje izvršnog fajla – build
  - `gcc -o hello hello.c`



Pretpocesiranje

hello.c

```
#include <stdio.h>
#include "hello.h"

#define max(a,b) (a)>(b)?(a):(b)

int main() {
    int x=3,y=4;
    printf("Hello, world!\n");
    printf("%d\n",max(x,y));
    return 0;
}
```

hello.h

```
void fun1(int,int);
```

- Uključivanje hedera (#include) i menjanje makroe (#define).

  `gcc -E hello.c > hello.i`

- `hello.i` sadrži prošireni izvorni kod.

Prevođenje u asembler

- Preprocesirani kod se prevodi u asemblerski kod specifičan za određenu familiju procesora.

```
gcc -S hello.i
```

hello.s

```
    .file               "hello.c"
    .section            .rodata
.LC0:    .string        "Hello, world!„
.LC1:    .string        "%d\n"
    .text
    .globl              main
    .type               main, @functionmain:
.LFB0:
    .cfi_startproc
    pushq               %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq %rsp, %rbp
    .cfi_def_cfa_register 6
    subq $16, %rsp
    movl $3, -8(%rbp)
    movl $4, -4(%rbp)
    movl $.LC0, %edi
    call puts
    movl -8(%rbp), %eax
    cmpl %eax, -4(%rbp)
    cmovge              -4(%rbp), %eax
    movl %eax, %esi
    movl $.LC1, %edi
    movl $0, %eax
    call printf
    movl $0, %eax
    leave               .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
    .size               main, .-main
    .ident              "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4)
    5.4.0 20160609„
    .section            .note.GNU-stack,"",@progbits
```

Prevođenje u mašinski

- Asembler (**as.exe**) prevodi asemblerski kod u mašinski i dobija se objektni fajl `hello.o`.

```
as -o hello.o hello.s
```

- Linker (**ld.exe**) povezuje objektni kod sa kodom korišćenih biblioteka i daje izvršni fajl.

```
ld -o apl obj.o ...libraries...
```

- Likovanje objektnih fajlova se u praksi češće izvodi komandom

```
gcc -o apl obj1.o obj2.o ...
```

Kompajliranje kalkulatora

- Čitav kod je raspoređen u datoteke

```
    main.c, stack.c, getop.c, getch.c, calc.h
gcc -c main.c
gcc -c stack.c
gcc -c getop.c
gcc -c getch.c
gcc -o calc main.o stack.o getop.o getch.o
```

- Redosled kompajliranja fajlova nije bitan
- Nakon izmena kompajliraju se samo fajlovi u kojima su izmene, napravljene i vrši se ponovno linkovanje

Make, CMake

- Olakšava kreiranje izvršnog koda od izvornih fajlova
- Pokretanje

  `make`

  `make clean`

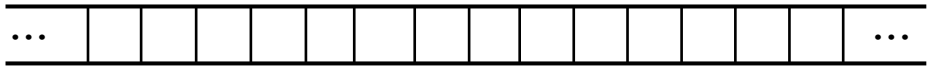
makefile

```
all: calc
calc:  main.o stack.o getop.o getch.o
	gcc -o calc main.o stack.o getop.o getch.o
main.o: main.c calc.h
	gcc -c main.c
stack.o:stack.c calc.h
	gcc -c stack.c
getop.o: getop.c calc.h
	gcc -c getop.c
getch.o: getch.c calc.h
	gcc -c getch.c

clean:
	rm -rf *o calc
```

POKAZIVAČI I NIZOVI

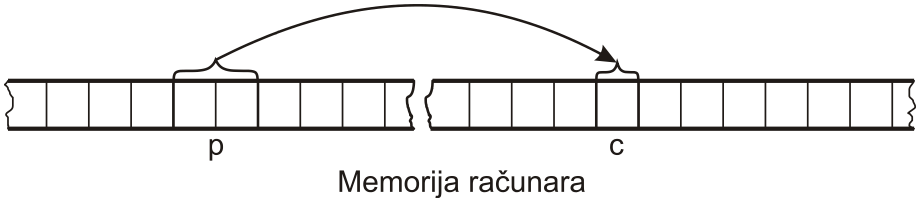
Pokazivači

- Pokazivač je promenljiva koja sadrži adresu druge promenljive
- Računar ima niz redom numerisanih (adresiranih) memorijskih ćelija kojima se može manipulisati pojedinačno ili u povezanim grupama
  - svaki bajt može predstavljati vrednost tipa `char`,
  - par jednobajtnih ćelija se može tretirati kao `short`
  - četiri uzastopna bajta formira vrednost tipa `long`



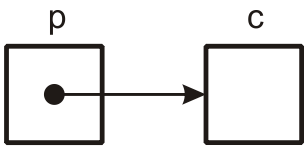
- Pokazivač je grupa ćelija (dve ili četiri) koja može sadržati adresu

Pokazivači i adrese



- Dobijanje adrese nekog objekta (operator &)

```
p = &c;    /* p pokazuje na c */
```



- Operator & se primenjuje samo na objekte u memoriji (ne na izraze, konstante, registarske promenljive)

Pokazivači i adrese

- Pristupanje objektu na koji pokazivač pokazuje – operator *, operator dereferenciranja

```
int x = 1, y = 2, z[10];
int *ip;  /* ip je pokazivac na int */

ip = &x;  /* ip sada pokazuje na x */
y = *ip;  /* y je sada 1 */
*ip = 0;  /* x je sada 0 */
ip = &z[3];      /* ip sada pokazuje na z[3] */
```

Pokazivači i adrese

- Deklaracija pokazivača

```
int *ip;
double *dp, atof(char *);
```

- Ako `ip` pokazuje na celobrojnu promenljivu, onda se `*ip` može koristiti u svakom kontekstu u kome se koristi ceo broj

```
*ip = *ip + 10;
```

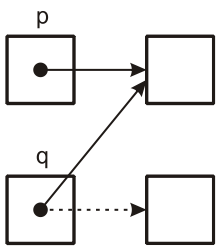
Pokazivači i adrese

- Unarni operatori & i * imaju viši prioritet od aritmetičkih operatora

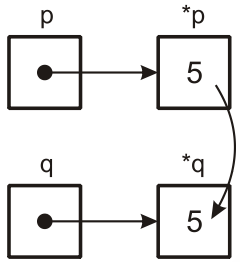
```
y = *ip + 1;
*ip += 1;
++*ip;
(*ip)++;
```

- Pokazivači su takođe promenljive, pa se mogu koristiti i bez dereferenciranja

```
q = p;
```




```
*q=*p;
```



Pokazivači i argumenti funkcija

- Pozvana funkcija ne može direktno da promeni promenljivu u funkciji koja je poziva.
- Argumenti funkcije – formalni parametri
  Promenljive koje prihvataju vrednosti koje su date funkciji pri pozivu.
- Primer: Funkcija koja menja mesta dvema promenljivama

```
swap(x,y);
...
void swap(int a, int b) /* POGRESNO*/
{
    int temp;

    temp = a;
    a = b;
    b = temp;
}
```

Gde su promenljive?

- Svakom procesu (program u izvršavanju) se dodeljuje određen segment radne memorije.

- STACK - (stek) je deo memorije koji je rezervisan za čuvanje lokalnih promenljivih, parametara procedura, povratnih adresa.

STEK

```
main(){
    int a;
    int b;
    a = 5;
    b = 10;
    swap(a,b);
    printf("%d %d",a,b);
}

void swap(int a, int b)
{
    int temp;

    temp = a;
    a = b;
    b = temp;
}
```

a

STEK

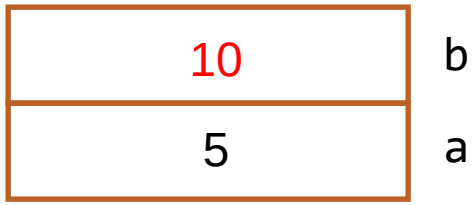
```
main(){
    int a;
    int b;
    a = 5;
    b = 10;
    swap(a,b);
    printf("%d %d",a,b);
}

void swap(int a, int b)
{
    int temp;

    temp = a;
    a = b;
    b = temp;
}
```

STEK

```
main(){
    int a;
    int b;
    a = 5;
    b = 10;
    swap(a,b);
    printf("%d %d",a,b);
}

void swap(int a, int b)
{
    int temp;

    temp = a;
    a = b;
    b = temp;
}
```

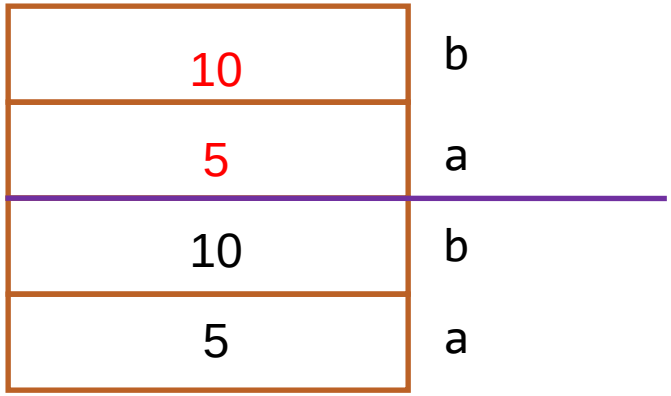
STEK

```
main(){
    int a;
    int b;
    a = 5;
    b = 10;
    swap(a,b);
    printf("%d %d",a,b);
}

void swap(int a, int b)
{
    int temp;

    temp = a;
    a = b;
    b = temp;
}
```



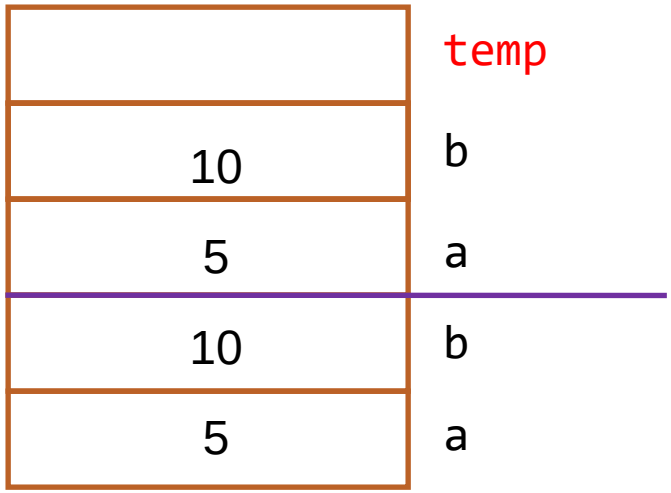
STEK

```
main(){
    int a;
    int b;
    a = 5;
    b = 10;
    swap(a,b);
    printf("%d %d",a,b);
}

void swap(int a, int b)
{
    int temp;

    temp = a;
    a = b;
    b = temp;
}
```



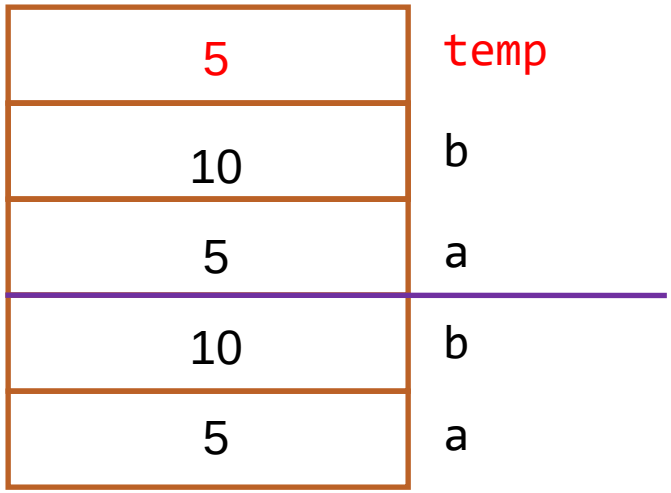
STEK

```
main(){
    int a;
    int b;
    a = 5;
    b = 10;
    swap(a,b);
    printf("%d %d",a,b);
}

void swap(int a, int b)
{
    int temp;

    temp = a;
    a = b;
    b = temp;
}
```



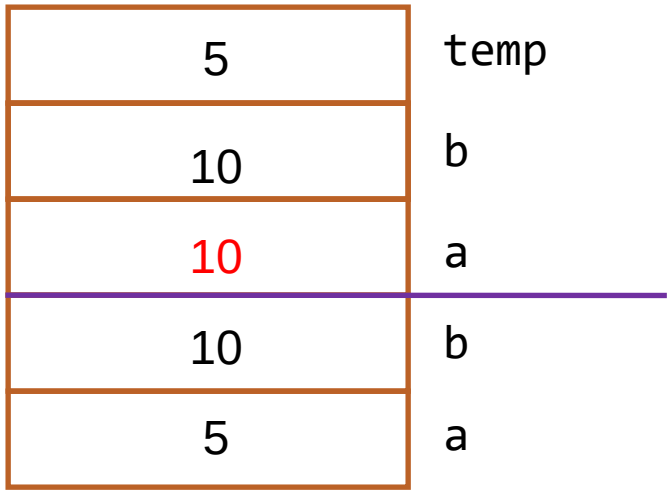
STEK

```
main(){
    int a;
    int b;
    a = 5;
    b = 10;
    swap(a,b);
    printf("%d %d",a,b);
}

void swap(int a, int b)
{
    int temp;

    temp = a;
    a = b;
    b = temp;
}
```



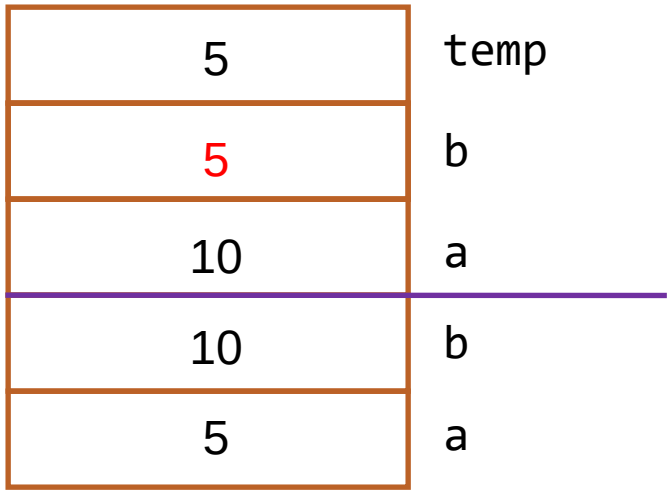
STEK

```
main(){
    int a;
    int b;
    a = 5;
    b = 10;
    swap(a,b);
    printf("%d %d",a,b);
}

void swap(int a, int b)
{
    int temp;

    temp = a;
    a = b;
    b = temp;
}
```



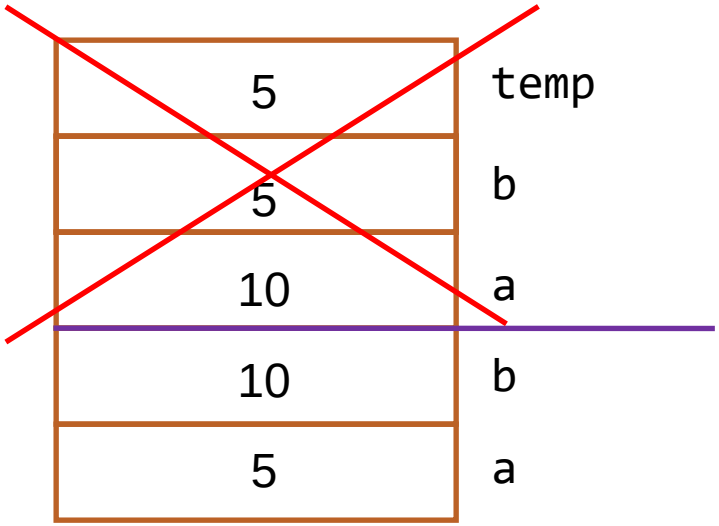
STEK

```
main(){
    int a;
    int b;
    a = 5;
    b = 10;
    swap(a,b);
    printf("%d %d",a,b);
}

void swap(int a, int b)
{
    int temp;

    temp = a;
    a = b;
    b = temp;
}
```



STEK

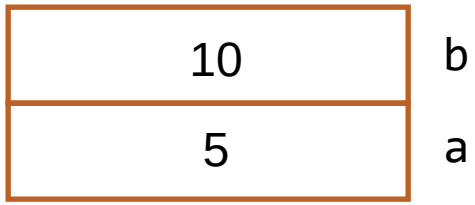
```
main(){
    int a;
    int b;
    a = 5;
    b = 10;
    swap(a,b);
    printf("%d %d",a,b);
}

void swap(int a, int b)
{
    int temp;

    temp = a;
    a = b;
    b = temp;
}
```



STEK

```
main(){
    int a;
    int b;
    a = 5;
    b = 10;
    swap(a,b);
    printf("%d %d",a,b);
}

void swap(int a, int b)
{
    int temp;

    temp = a;
    a = b;
    b = temp;
}
```

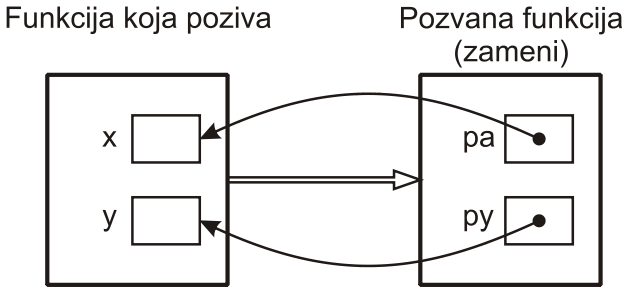
→ 5 10



Pokazivači i argumenti funkcija

- Promena se može postići slanjem pokazivača na promenljive koje treba promeniti

```
swap(&x, &y);
...

void swap(int *pa, int *pb) /* zamena *pa i *pb */
{
    int temp;

    temp = *pa;
    *pa = *pb;
    *pb = temp;
}
```



Shematski prikaz prosleđivanja pokazivača funkciji

STEK

```
main(){
    int a;
    int b;
    a = 5;
    b = 10;
    swap(&a,&b);
    printf("%d %d",a,b);
}

void swap(int *a, int *b)
{
    int temp;

    temp = *a;
    *a = *b;
    *b = temp;
}
```

a

STEK

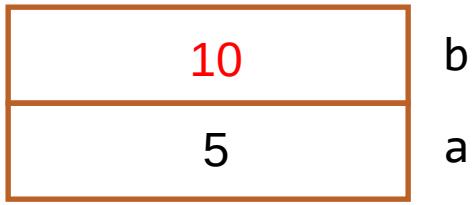
```
main(){
    int a;
    int b;
    a = 5;
    b = 10;
    swap(&a,&b);
    printf("%d %d",a,b);
}

void swap(int *a, int *b)
{
    int temp;

    temp = *a;
    *a = *b;
    *b = temp;
}
```

STEK

```
main(){
    int a;
    int b;
    a = 5;
    b = 10;
    swap(&a,&b);
    printf("%d %d",a,b);
}

void swap(int *a, int *b)
{
    int temp;

    temp = *a;
    *a = *b;
    *b = temp;
}
```

STEK

```
main(){
    int a;
    int b;
    a = 5;
    b = 10;
    swap(&a,&b);
    printf("%d %d",a,b);
}

void swap(int *a, int *b)
{
    int temp;

    temp = *a;
    *a = *b;
    *b = temp;
}
```

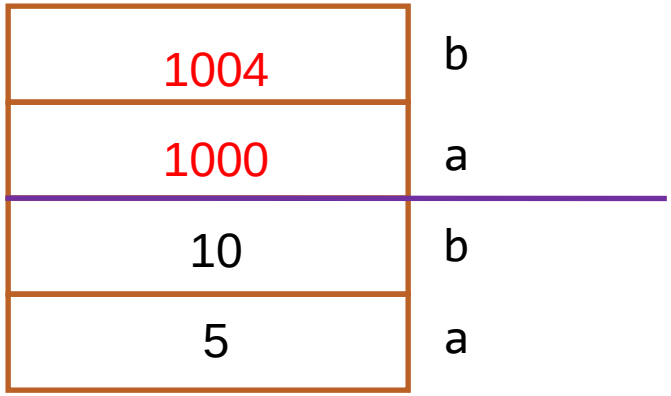


STEK

```
main(){
    int a;
    int b;
    a = 5;
    b = 10;
    swap(&a,&b);
    printf("%d %d",a,b);
}

void swap(int *a, int *b)
{
    int temp;

    temp = *a;
    *a = *b;
    *b = temp;
}
```

&b - 1004

&a - 1000

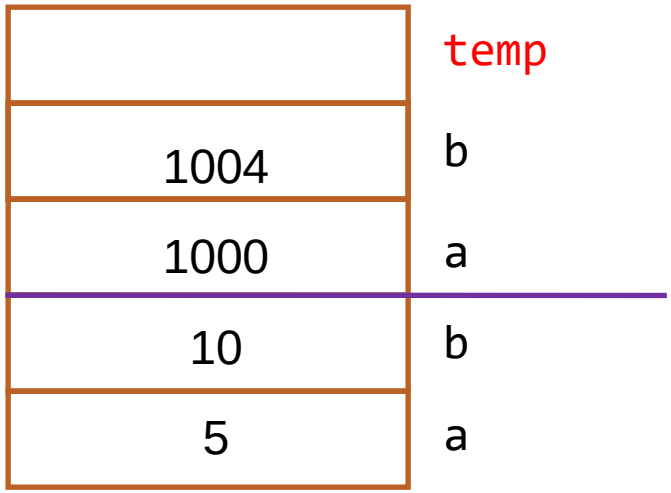


STEK

```
main(){
    int a;
    int b;
    a = 5;
    b = 10;
    swap(&a,&b);
    printf("%d %d",a,b);
}

void swap(int *a, int *b)
{
    int temp;

    temp = *a;
    *a = *b;
    *b = temp;
}
```

&b - 1004

&a - 1000

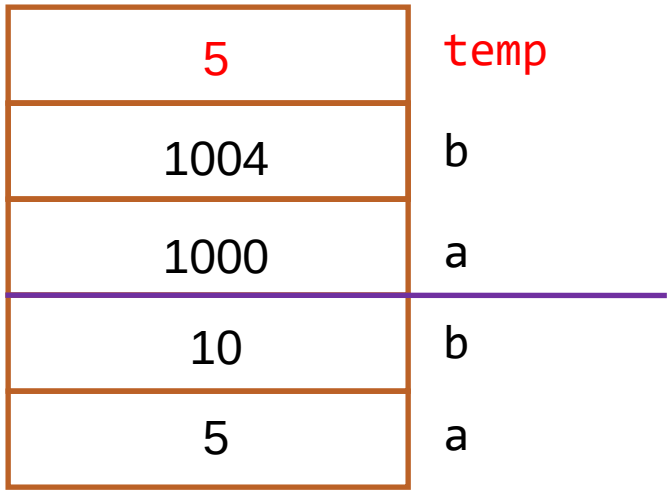


STEK

```
main(){
    int a;
    int b;
    a = 5;
    b = 10;
    swap(&a,&b);
    printf("%d %d",a,b);
}

void swap(int *a, int *b)
{
    int temp;

    temp = *a;
    *a = *b;
    *b = temp;
}
```

&b - 1004

&a - 1000

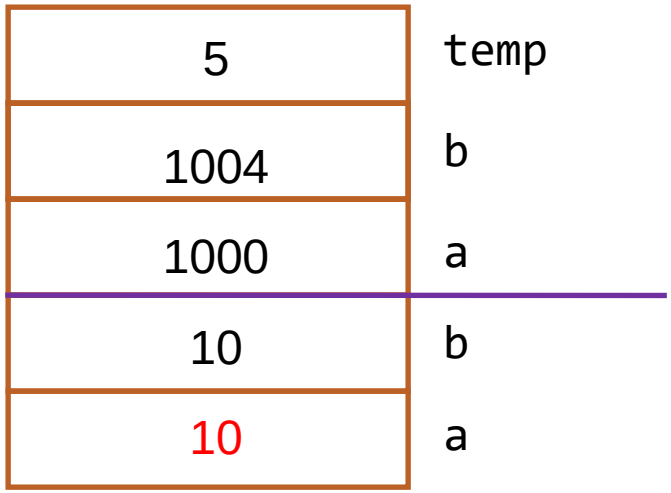


STEK

```
main(){
    int a;
    int b;
    a = 5;
    b = 10;
    swap(&a,&b);
    printf("%d %d",a,b);
}

void swap(int *a, int *b)
{
    int temp;

    temp = *a;
    *a = *b;
    *b = temp;
}
```

&b - 1004

&a - 1000

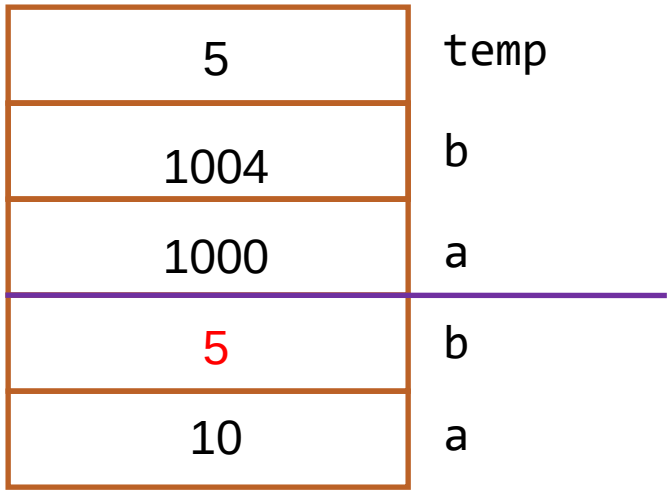


STEK

```
main(){
    int a;
    int b;
    a = 5;
    b = 10;
    swap(&a,&b);
    printf("%d %d",a,b);
}

void swap(int *a, int *b)
{
    int temp;

    temp = *a;
    *a = *b;
    *b = temp;
}
```

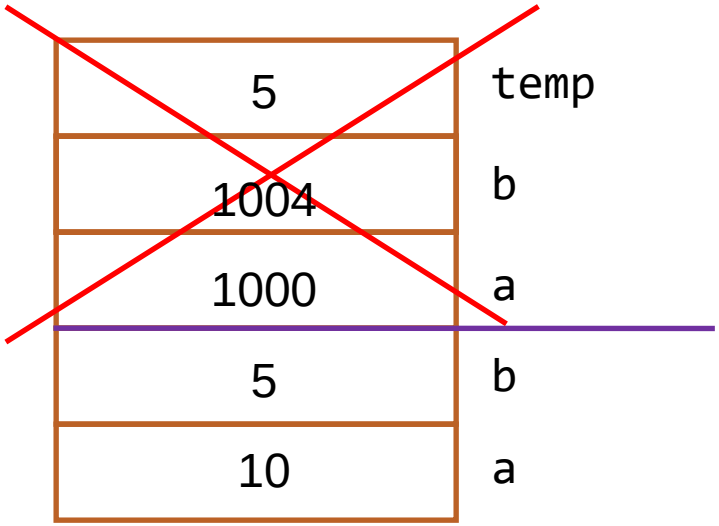
&b - 1004

&a - 1000



STEK

```
main(){
    int a;
    int b;
    a = 5;
    b = 10;
    swap(&a,&b);
    printf("%d %d",a,b);
}

void swap(int *a, int *b)
{
    int temp;

    temp = *a;
    *a = *b;
    *b = temp;
}
```



STEK

```
main(){
    int a;
    int b;
    a = 5;
    b = 10;
    swap(&a,&b);
    printf("%d %d",a,b);
}

void swap(int *a, int *b)
{
    int temp;

    temp = *a;
    *a = *b;
    *b = temp;
}
```

→ 10 5