arrays_strings_pointers.doc

NOTA: estão são alguns dos exemplos apresentados na aula teórica
os comentários ao código são reduzidos; foram feitos na aula

```
//main1.c – DECLARACAO DE ARRAYS + LEITURA DE STRINGS

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

#define NOME_MAX_COMP 50 //evitar "magic numbers"
#define NUM_MESES 12

int main(int argc, char *argv[])
{
	char nome[NOME_MAX_COMP];
	int peso[NUM_MESES];
	int i;

	printf ("Nome ? ");
	scanf("%s",nome); //tentar c/ "Rui Santos"

	for (i=0; i<NUM_MESES; i++)
	{
		printf("peso[%d] ? ",i); scanf("%d",&peso[i]);
	}
	for (i=0; i<NUM_MESES; i++)
	{
		printf("peso[%d] = %d\n",i,peso[i]);
	}

	// CÓDIGO A COMPLETAR C/ PROCESSAM. DOS DADOS ...

	return 0;
}
```

```c
//main2.c – DECLARACAO DE ARRAYS + LEITURAS DO TECLADO (CUIDADOS)

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define NOME_MAX_COMP 10 //NOTAR
#define NUM_MESES 4

int main(int argc, char *argv[])
{
	char nome[NOME_MAX_COMP];
	int peso[NUM_MESES];
	int i;

	printf ("Nome ? ");
	fgets(nome,sizeof(nome),stdin); // por que não gets() ?
	// EXPERIMENTAR CORRER COM NOME1 =
	// 1) 123
	// 2) 1234567890
	// 3) 123456789
	// E INTERPRETAR OS RESULTADOS
	// (VERIFICAR QUE EM ALGUNS CASOS 'NOME' JA´ INCLUI O NEWLINE)

	/* TESTAR S/ e C/ ESTE CÓDIGO
	if ((strlen(nome)==(sizeof(nome)-1)) && nome[strlen(nome)-1]!='\n')
		while(getchar()!='\n');
	*/

	for (i=0; i<NUM_MESES; i++)
	{
		printf("peso[%d] ? ",i); scanf("%d",&peso[i]);
		// perror("main"); // NAO DA' ERRO; testar valor de retorno de
scanf()
	}

	// mostrar valores lidos
	printf("\n\n%s\n",nome);
	for (i=0; i<NUM_MESES; i++)
	{
		printf("peso[%d] = %d\n",i,peso[i]);
	}

	return 0;
}
```

```c
//main3.c -  STRINGS DE C

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_LEN 10

int main(int argc, char *argv[])
{
      // Declaração de strings
      char nome1[MAX_LEN] = "Ana Sousa";
      char nome2[MAX_LEN];
      char *nome3; // ATRIBUIDO A CONSTANTE (V. ABAIXO)
      char *nome4;
      char *nome5=NULL;

      printf("%s\n",nome1);
      // nome2 = "Rui Santos"; // TESTAR

      nome3="Pedro Silva";
      printf("%s\n",nome3);

      nome4=(char *) malloc(MAX_LEN*sizeof(char));
      printf("Nome4 ? "); fgets(nome4,MAX_LEN,stdin);
      printf("\nnome4 = %s\n",nome4);

      // QUAL A DIFERENCA ENTRE AS 3 SEQ.S DE INSTRUCOES SEGUINTES ?

      //1) -
      //nome5=nome4;

      //2) -
      // strcpy(nome5,nome4); // sintaxe: strcpy(destination,source)

      //3) - CORRIGE O ERRO DE 2)
      //nome5=(char *) malloc(MAX_LEN*sizeof(char));
      //strcpy(nome5,nome4); //strcpy(destination,source)

      //4)
      nome5 = &nome4[1];

      printf("nome5 = %s\n",nome5); // imprime "SOUSA"

      return 0;
}
```

```c
//main4.c -   // RELAÇÃO ENTRE ARRAYS E APONTADORES

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_LEN 10

int main(int argc, char *argv[])
{
      int i;
      int a[MAX_LEN];
      int *b;
      int c[MAX_LEN];

      for (i=0; i<MAX_LEN; i++)
            a[i]=i;

      b = a;
      // c = a;  //TESTAR (o que acontece ?)

      for (i=0; i<MAX_LEN; i++)
            printf ("a[%d]=%d; b[%d]=%d\n",i, a[i], i, b[i]);

      printf("\n");

      for (i=0; i<MAX_LEN; i++)
            printf ("b[%d]=%d\n",i,*(b+i));

      return 0;
}
```

```c
//main5a.c - // ARRAYS DE STRINGS

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_LEN 10

void mostra(int n,char nms[][MAX_LEN])
{
     int i;
     for (i=0;i<n;i++)
          printf("%s\n",nms[i]);
}

int main(int argc, char *argv[])
{
     char nomes[3][MAX_LEN];

     strcpy(nomes[0],"Ana Sousa");
     strcpy(nomes[1],"Rui Silva");
     mostra(2,nomes);

     return 0;
}
```

```c
//main5b.c - // ARRAYS DE STRINGS

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_LEN 10

// POR QUE DA' ERRO DE COMPILACAO ?
void mostra(int n, char *nms[])
{
     int i;
     for (i=0;i<n;i++)
          printf("%s\n",nms[i]);
}

int main(int argc, char *argv[])
{
     char nomes[3][MAX_LEN];

     strcpy(nomes[0],"Ana Sousa");
     strcpy(nomes[1],"Rui Silva");
     mostra(2,nomes);

     system("PAUSE");
     return 0;
}
```

```c
//main6a.c -  ARRAYS DE STRINGS

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_LEN 10
#define MAX_NOMES 3  // >= 2

void mostra(int n,char *nms[])
// ALTERNATIVA: void mostra(int n,char **nms)
{
      int i;
      for (i=0;i<n;i++)
            printf("%s\n",nms[i]);
}

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

      int i;

      // reserva dinamicamente a memoria
      for (i=0;i<MAX_NOMES;i++)
            nomes[i]=(char *) malloc(MAX_LEN*sizeof(char));

      // actualiza array de nomes
      strcpy(nomes[0],"Ana Sousa");
      strcpy(nomes[1],"Rui Silva");
      mostra(2,nomes);

      // liberta a memoria reservada dinamicamente
      for (i=0; i<MAX_NOMES; i++)
            free(nomes[i]);

      // mostra(2,nomes);  // CUIDADO!, a memória já foi libertada

      return 0;
}
```

```c
//main6b.c -  ARRAYS DE STRINGS

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_LEN 10

void lerNomes(int n, char **nms)
{
	int i;

	for (i = 0; i < n; i++)
	{
		printf("Nome [%d] ? ", i);
		fgets(nms[i],MAX_LEN,stdin);
	}
}


void mostrarNomes(int n,char **nms)
{
	int i;
	for (i=0;i<n;i++)
		printf("%s",nms[i]);
}

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

	int i, n;

	printf("Quantos nomes ? ");
	scanf("%d",&n);
	while (getchar() !='\n'); //fflush(stdin);

	// reserva dinamicamente a memoria
	nomes = (char **) malloc(n*sizeof(char *));
	for (i = 0; i < n; i++)
		nomes[i] = (char *) malloc(MAX_LEN*sizeof(char));

	// le^ os nomes
	lerNomes(n,nomes);

	// mostra os nomes
	mostrarNomes(n,nomes);

	// liberta a memoria reservada dinamicamente
	for (i = 0; i < n; i++)
		free(nomes[i]);
	free(nomes);

	return 0;
}
```

```c
//main7.c - FUNCOES E APONTADORES

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_LEN 10

void teste1(void)
{
     printf("Hello\n");
}

void teste2(int n)
{
     int i;
     for (i=0; i<n; i++)
          printf("Hello no. %d\n",i);

     printf("Hello\n");
}

int main(int argc, char *argv[])
{
     void (*func1) (void);
     void (*func2) (int);

     func1 = teste1;
     func1();

     func2 = teste2;
     func2(3);

     return 0;
}
```

```c
/* LEITURA DE PASSWORD, SEM ECOAR CARACTER LIDO */

#include <termios.h>
#include <unistd.h>
#include <string.h>

#define MAX_PASSWD_LEN 20

int main(void)
{
 struct termios term, oldterm;
 int i;
 char pass[MAX_PASSWD_LEN+1], ch, echo = '*';

 write(STDOUT_FILENO, "\nPassword? ", 11);

 tcgetattr(STDIN_FILENO, &oldterm);
 term = oldterm;
 term.c_lflag &= ~(ECHO | ECHOE | ECHOK | ECHONL | ICANON);
 tcsetattr(STDIN_FILENO, TCSAFLUSH, &term);

 i=0;
 while (i < MAX_PASSWD_LEN && read(STDIN_FILENO, &ch, 1) &&
        ch != '\n') {
  pass[i++] = ch;
  write(STDOUT_FILENO, &echo, 1);
 }
 pass[i] = 0;

 tcsetattr(STDIN_FILENO, TCSANOW, &oldterm);

 write(STDOUT_FILENO, "\n\nPassword: ", 12);
 write(STDOUT_FILENO, pass, strlen(pass));
 write(STDOUT_FILENO, "\n", 1);

 return 0;
}

/* RESULTADOS DE EXECUÇÃO:
pinguim> readpass

Password? *****

Password: 12345
pinguim> readpass

Password? ********************

Password: 12345678901234567890          <- atingiu 20 caracteres
```

```c
/* COPIA FICHEIRO */
/* USO: copy source destination */

#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>

#define BUFFER_SIZE 512

int main(int argc, char *argv[])
{
 int fd1, fd2, nr, nw;
 unsigned char buffer[BUFFER_SIZE];

 if (argc != 3) {
  printf("Usage: %s <source> <destination>\n", argv[0]);
  return 1;
 }
 fd1 = open(argv[1], O_RDONLY);
 if (fd1 == -1) {
  perror(argv[1]);
  return 2;
 }
 fd2 = open(argv[2], O_WRONLY | O_CREAT | O_EXCL, 0644);
 if (fd2 == -1) {
  perror(argv[2]);
  close(fd1);
  return 3;
 }
 while ((nr = read(fd1, buffer, BUFFER_SIZE)) > 0)
  if ((nw = write(fd2, buffer, nr)) <= 0 || nw != nr) {
   perror(argv[2]);
   close(fd1);
   close(fd2);
   return 4;
  }
  close(fd1);
  close(fd2);
  return 0;
}
```

```c
/* COPIA FICHEIRO p/ECRAN OU p/OUTRO FICHEIRO */
/* USO:
   - MOSTRAR NO ÉCRAN --------> copy source
   - COPIAR P/OUTRO FICHEIRO -> copy source destination
*/

#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>

#define BUFFER_SIZE 512

int main(int argc, char *argv[])
{
 int fd1, fd2, nr, nw;
 unsigned char buffer[BUFFER_SIZE];

 if ((argc != 2) && (argc != 3)) {
  printf("Usage: %s <source>  OR  %s <source> <destination>\n",
         argv[0], argv[0]);
  return 1;
 }
 fd1 = open(argv[1], O_RDONLY);
 if (fd1 == -1) {
  perror(argv[1]);
  return 2;
 }
 if (argc == 3) {
  fd2 = open(argv[2], O_WRONLY | O_CREAT | O_EXCL, 0644);
  if (fd2 == -1) {
   perror(argv[2]);
   close(fd1);
   return 3;
  }
  dup2(fd2,STDOUT_FILENO);
 }
 while ((nr = read(fd1, buffer, BUFFER_SIZE)) > 0)
  if ((nw = write(STDOUT_FILENO, buffer, nr)) <= 0 || nw != nr) {
   perror(argv[2]);
   close(fd1);
   close(fd2);
   return 4;
  }
  close(fd1);
  if (argc == 3)
   close(fd2);
  return 0;
}
```

```c
/* LISTAR FICHEIROS REGULARES DE UM DIRECTÓRIO */
/* USO: listdir dirname */

#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <dirent.h>
#include <errno.h>
#include <string.h>

int main(int argc, char *argv[])
{
 DIR *dir;
 int line;
 struct dirent *dentry;
 struct stat stat_entry;

 if (argc != 2) {
  printf("Usage: %s <dir_path>\n", argv[0]);
  return 1;
 }
 if ((dir = opendir(argv[1])) == NULL) {
  perror(argv[1]);
  return 2;
 }

 chdir(argv[1]);

 printf("Ficheiros regulares do directorio '%s'\n", argv[1]);
 line = 1;
 while ((dentry = readdir(dir)) != NULL) {
  stat(dentry->d_name, &stat_entry);
  if (S_ISREG(stat_entry.st_mode)) {
   printf("%-25s%12d%3d\n", dentry->d_name,
          (int)stat_entry.st_size,(int)stat_entry.st_nlink);
   if (line++ % 20 == 0) {
     printf("Press <enter> to continue");
     getchar();
   }
  }
 }
 return 0;
}
```

```
/* RESULTADO DE EXECUÇÃO:

pinguim> listdir .
Ficheiros regulares do directorio '.'
p1.c                                754  1
p2.c                                795  1
p3.c                                940  1
p4a.c                               909  1
p4b.c                              1050  1
p1                                11307  1
p2                                11484  1
p4a                               11739  1
p4b                               11934  1
p2.txt                              795  1
p3                                11596  1
p3.txt                              940  1
p1a.c                               810  1
p1a                               11308  1
pinguim> ln p2.txt p2link
pinguim> listdir .
Ficheiros regulares do directorio '.'
p1.c                                754  1
p2.c                                795  1
p3.c                                940  1
p4a.c                               909  1
p4b.c                              1050  1
p1                                11307  1
p2                                11484  1
p4a                               11739  1
p4b                               11934  1
p2.txt                              795  2          <--- NOTAR
p3                                11596  1
p3.txt                              940  1
p1a.c                               810  1
p1a                               11308  1
p2link                              795  2          <--- NOTAR
pinguim>
*/
```

```c
/* LISTAR FICHEIROS REGULARES E SUB-DIRECTÓRIOS DE UM DIRECTÓRIO */
/* USO: listdir2 dirname */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <sys/stat.h>
#include <errno.h>

int main(int argc, char *argv[])
{
 DIR *dirp;
 struct dirent *direntp;
 struct stat stat_buf;
 char *str;
 char name[200];

 if (argc != 2)
 {
  fprintf( stderr, "Usage: %s dir_name\n", argv[0]);
  exit(1);
 }

 if ((dirp = opendir( argv[1])) == NULL)
 {
  perror(argv[1]);
  exit(2);
 }

 while ((direntp = readdir( dirp)) != NULL)
 {
  sprintf(name,"%s/%s",argv[1],direntp->d_name); // <----- NOTAR
                          // alternativa a chdir(); ex: anterior
  if (lstat(name, &stat_buf)==-1)   // testar com stat()
  {
   perror("lstat ERROR");
   exit(3);
  }
  //      printf("%10d - ",(int) stat_buf.st_ino);
  if (S_ISREG(stat_buf.st_mode)) str = "regular";
  else if (S_ISDIR(stat_buf.st_mode)) str = "directory";
  else str = "other";
  printf("%-25s - %s\n", direntp->d_name, str);
 }

 closedir(dirp);
 exit(0);
}
```

```
/* RESULTADOS :


pinguim> gcc listdir2.c -o listdir2 -Wall     <--- programa que usa lstat()
pinguim> listdir2 ..

pinguim> ls .. -lai
total 36
 768201 drwxrwxr-x    7 ... ...   4096 Sep 26 23:47 .
 670454 drwxrwxr-x    6 ... ...   4096 Sep 21 23:12 ..
 198097 drwxrwxr-x    2 ... ...   4096 Sep 26 22:09 d1
 198121 drwxrwxr-x    3 ... ...   4096 Sep 26 22:16 d2
 165546 drwxrwxr-x    2 ... ...   4096 Sep 26 22:23 examples
 768210 -rw-rw-r--    1 ... ...   3329 Sep 26 22:24 ln.txt
 654158 drwxrwxr-x    2 ... ...   4096 Sep 26 23:49 probs_01
 768202 drwxrwxr-x    2 ... ...   4096 Sep 26 17:02 probs_02
 198122 -rw-------    2 ... ...    912 Sep 26 17:03 t1.c                  <---HARD LINK
 768211 lrwxrwxrwx    1 ... ...     11 Sep 26 23:47 t2.c -> d2/f2_1.txt   <---SYMBOLIC LINK

pinguim> listdir2 ..
.                       - directory
..                      - directory
t1.c                    - regular
probs_02                - directory
d1                      - directory
d2                      - directory
examples                - directory
ln.txt                  - regular
probs_01                - directory
t2.c                    - other        <--- NOTAR



pinguim> gcc listdir2.c -o listdir2-Wall    <---  usando stat(), em vez de lstat()

pinguim> listdir2 ..
.                       - directory
..                      - directory
t1.c                    - regular     <--- NOTAR
probs_02                - directory
d1                      - directory
d2                      - directory
examples                - directory
ln.txt                  - regular
probs_01                - directory
t2.c                    - regular     <--- NOTAR: o fich. apontado, d2/f2_1.txt, é "regular"

pinguim>


*/
```

HARD LINKS & SYMBOLIC LINKS

```
pinguim> ls
d1  d2  Probs_01  Probs_02

pinguim> ls -lai
total 24
 768201 drwxrwxr-x    6   ... ...   4096 Sep 26 22:07 .
 670454 drwxrwxr-x    6   ... ...   4096 Sep 21 23:12 ..
 198097 drwxrwxr-x    2   ... ...   4096 Sep 26 22:09 d1
 198121 drwxrwxr-x    3   ... ...   4096 Sep 26 22:16 d2
 654158 drwxrwxr-x    2   ... ...   4096 Sep 26 17:01 Probs_01
 768202 drwxrwxr-x    2   ... ...   4096 Sep 26 17:02 Probs_02

pinguim> ls d1 -lai
total 20
 198097 drwxrwxr-x    2   ... ...   4096 Sep 26 22:09 .
 768201 drwxrwxr-x    6   ... ...   4096 Sep 26 22:07 ..
 198122 -rw-------    1   ... ...    912 Sep 26 17:03 p1_1.c
 198123 -rw-------    1   ... ...    795 Sep 26 17:03 p1_2.c
 198124 -rw-------    1   ... ...    956 Sep 26 17:03 p1_3.c

pinguim> ls d2 -lai
total 20
 198121 drwxrwxr-x    3   ... ...   4096 Sep 26 22:16 .
 768201 drwxrwxr-x    6   ... ...   4096 Sep 26 22:07 ..
  51524 drwxrwxr-x    2   ... ...   4096 Sep 26 22:18 d2_1
 198127 -rw-------    1   ... ...    912 Sep 26 17:07 f2_1.txt
 198128 -rw-------    1   ... ...    795 Sep 26 17:07 f2_2.txt
```

```
pinguim> ln d1/p1_1.c t1.c

pinguim> ln -s d2/f2_2.txt t2.txt

pinguim> ls -lai
total 28
 768201 drwxrwxr-x    6   ... ...   4096 Sep 26 22:21 .
 670454 drwxrwxr-x    6   ... ...   4096 Sep 21 23:12 ..
 198097 drwxrwxr-x    2   ... ...   4096 Sep 26 22:09 d1
 198121 drwxrwxr-x    3   ... ...   4096 Sep 26 22:16 d2
 654158 drwxrwxr-x    2   ... ...   4096 Sep 26 17:01 Probs_01
 768202 drwxrwxr-x    2   .....   4096 Sep 26 17:02 Probs_02
 198122 -rw-------    2   ... ...    912 Sep 26 17:03 t1.c
 768210 lrwxrwxrwx    1   ... ...     11 Sep 26 22:21 t2.txt -> d2/f2_2.txt

pinguim> ls d1 -lai
total 20
 198097 drwxrwxr-x    2   ... ...   4096 Sep 26 22:09 .
 768201 drwxrwxr-x    6   ... ...   4096 Sep 26 22:21 ..
 198122 -rw-------    2   ... ...    912 Sep 26 17:03 p1_1.c
 198123 -rw-------    1   ... ...    795 Sep 26 17:03 p1_2.c
 198124 -rw-------    1   ... ...    956 Sep 26 17:03 p1_3.c

pinguim> ls d2 -lai
total 20
 198121 drwxrwxr-x    3   ... ...   4096 Sep 26 22:16 .
 768201 drwxrwxr-x    6   ... ...   4096 Sep 26 22:21 ..
  51524 drwxrwxr-x    2   ... ...   4096 Sep 26 22:18 d2_1
 198127 -rw-------    1   ... ...    912 Sep 26 17:07 f2_1.txt
 198128 -rw-------    1   ... ...    795 Sep 26 17:07 f2_2.txt

pinguim> rm t1.c
pinguim> rm t2.txt

pinguim> ls -lai
total 24
 768201 drwxrwxr-x    6   ... ...   4096 Sep 26 22:22 .
 670454 drwxrwxr-x    6   ... ...   4096 Sep 21 23:12 ..
 198097 drwxrwxr-x    2   ... ...   4096 Sep 26 22:09 d1
 198121 drwxrwxr-x    3   ... ...   4096 Sep 26 22:16 d2
 654158 drwxrwxr-x    2   ... ...   4096 Sep 26 17:01 Probs_01
 768202 drwxrwxr-x    2   ... ...   4096 Sep 26 17:02 Probs_02
[jsilva@tintin 2005-06]$
```

FORK – EXEC – SYSTEM

```c
//=========================================================================
// f01.c / JAS
// Fork return value is different for 'parent' and 'child'
// Who is the 'parent' of the 'parent' ? Execute 'ps' command to see ...
//-------------------------------------------------------------------------

#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main(void)
{
  int fork_return_value;

  printf("before fork...\n");

  fork_return_value=fork();

  printf("I'm process %d: 'fork_return_value'=%d.
          My parent is %d.\n\n",getpid(),fork_return_value,getppid());

  return 0;
}
```

```c
//========================================================================
// f02.c / JAS
// Father & son. Which one runs first, after fork() ?
// Run several times and interpret results
//------------------------------------------------------------------------

#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main(void)
{
  int pid;

  printf("before fork...\n"); // remove '\n' and see what happens

  pid=fork();

  if (pid > 0)
    printf("I'm the parent (PID=%d)\n\n",getpid());
  else
    printf("I'm the son (PID=%d)\n\n",getpid());
  printf ("PID=%d exiting ...\n",getpid());

  return 0;
}


//========================================================================
// f02a.c / JAS
// Fork & output buffering
// Equal to f02.c with '\n' remove in "before fork ..." message
//------------------------------------------------------------------------

#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main(void)
{
  int pid;

  printf("before fork..."); // '\n' was removed

  pid=fork();

  if (pid > 0)
    printf("I'm the parent (PID=%d)\n\n",getpid());
  else
    printf("I'm the son (PID=%d)\n\n",getpid());
  printf ("PID=%d exiting ...\n",getpid());

  return 0;
}
```

```c
//=========================================================================
// f03.c / JAS
// Fork & output buffering
// Equal to f02.c print("before fork ...") replaced by write(...)
//-------------------------------------------------------------------------

#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main(void)
{
  int pid;

  write(STDOUT_FILENO,"before fork...",14); // printf() replaced by write()

  pid=fork();

  if (pid > 0)
    printf("I'm the parent (PID=%d)\n\n",getpid());
  else
    printf("I'm the son (PID=%d)\n\n",getpid());
  printf ("PID=%d exiting ...\n",getpid());

  return 0;
}
```

```c
//=========================================================================
// f04.c / JAS
// Basic synchronization. Father waits for the son to end.
//-------------------------------------------------------------------------

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(void)
{
  pid_t pid, pidSon;
  int status;

  pid=fork();
  if (pid > 0) {
    pidSon = wait(&status);
    printf("I'm the parent (PID=%d)\n\n",getpid());
    printf("My son %d exited with exit code %d\n",
           pidSon, WEXITSTATUS(status));}
  else
  {
    printf("I'm the son (PID=%d)\n\n",getpid());
    exit( getpid() % 10 );
  }
  printf ("PID=%d exiting ...\n",getpid());
  return 0;
}
```

```c
//=========================================================================
// f05.c / JAS
// zombie's
// In another terminal, execute command 'ps u'
//-------------------------------------------------------------------------

#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main(void)
{
  int pid;

  pid=fork();
  if (pid > 0) {
    printf("I'm the parent (PID=%d)\n\n",getpid());
    sleep(10); }
  else {
    printf("I'm the son (PID=%d)\n\n",getpid());
  }
  printf ("PID=%d exiting ...\n",getpid());
  return 0;
}
```

```c
//=========================================================================
// f06.c / JAS
// Tree of child processes with some zombies
// In another terminal, execute command 'ps u'
//-------------------------------------------------------------------------

#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main(void)
{
 int i, pid;

 for (i=1;i<=3;i++) {
  pid=fork();
  if (pid > 0) {
   printf("I'm the parent (PID=%d)\n\n",getpid());
   sleep(5);
  }
  else {
   printf("I'm the son (PID=%d). My parent is %d\n\n",getpid(),getppid());
   break;  // NOTE THIS
  }
 }
 printf ("PID=%d exiting ...\n",getpid());
 return 0;
}
```

```c
//=========================================================================
// e01.c / JAS
// execl() & execlp()
//-------------------------------------------------------------------------

#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(void)
{
  int pid;

  pid=fork();
  if (pid > 0) {
    wait(NULL); //father does not care w/exit status of the son ...
    printf("I'm the parent (PID=%d)\n\n",getpid()); }
  else {
    printf("I'm the son (PID=%d)\n\n",getpid());
    execl("ls","ls","-la",NULL); //try with execlp()
    printf(".... \n"); //which message makes sense, here ?
  }
  printf ("PID=%d exiting ...\n",getpid());
  return 0;
}
```

```c
//=========================================================================
// e01.c / JAS
// execl() & execlp()
//-------------------------------------------------------------------------

#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(void)
{
  int pid;
  int status;

  pid=fork();
  if (pid > 0) {
    wait(&status);
    printf("I'm the parent (PID=%d)\n\n",getpid());
    printf("My son exited with EXIT CODE = %d\n",WEXITSTATUS(status));}
  else {
    printf("I'm the son (PID=%d)\n\n",getpid());
    execlp("ls","ls","-la",NULL); //try with execl()
    //execl("./e01_aux","e01_aux","3",NULL);
    printf(".... \n"); //which message makes sense, here ?
  }
  printf ("PID=%d exiting ...\n",getpid());
  return 0;
}
```

```c
//=========================================================================
// e01_aux.c / JAS
// To be executed with e01.c
//-------------------------------------------------------------------------

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

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

  n = atoi(argv[1]);
  for (i=1; i<=n; i++)
  {
    printf("CHILD  (%d - %d): Hello father ...%d!\n",getpid(),getppid(),i);
  }

  return 10;
}
```

```c
//=========================================================================
// e02.c / JAS
// exec()
//-------------------------------------------------------------------------

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(void)
{
  int pid;
  int status;

  pid=fork();
  if (pid > 0) {
    wait(&status);
    printf("I'm the parent (PID=%d)\n\n",getpid());
    printf("My son exited with EXIT CODE = %d\n",WEXITSTATUS(status));}
  else {
    printf("I'm the son (PID=%d)\n\n",getpid());
    execlp("cat","cat","e02.c",NULL); // change "e02.c" to "xxxxx.c"
    printf("exec() failed !!! \n");
    exit(1);
  }
  printf ("PID=%d exiting ...\n",getpid());
  return 0;
}
```

```c
//=========================================================================
// e03.c / JAS
// exec()
//-------------------------------------------------------------------------

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(void)
{
  int pid;
  int status;

  pid=fork();
  if (pid > 0) {
    wait(&status);
    printf("I'm the parent (PID=%d)\n\n",getpid());
    printf("My son exited with EXIT CODE = %d\n",WEXITSTATUS(status));}
  else {
    printf("I'm the son (PID=%d)\n\n",getpid());
    execlp("cat","cat","e03.c",">","e03_copy.c",NULL);
    // note the "no such file or directory" errors of "cat"...!
    printf("exec() failed !!! \n");
    exit(1);
  }
  printf ("PID=%d exiting ...\n",getpid());
  return 0;
}
```

```c
//=========================================================================
// e04.c / JAS
// exec()
//-------------------------------------------------------------------------

#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>


int main(void)
{
  int pid;
  int status;
  char *arg[]={"ls","-laR",NULL};

  printf("before fork\n");
  pid=fork();
  if (pid > 0) {
    wait(&status);
    printf("I'm the parent (PID=%d)\n\n",getpid()); }
  else {
    printf("I'm the son (PID=%d)\n\n",getpid());
    execvp("ls",arg);
    printf("EXEC failed\n");
  }
  printf ("PID=%d exiting ...\n",getpid());
  return 0;
}
```

```c
//=========================================================================
// e05.c / JAS
// A simple command interpreter
//-------------------------------------------------------------------------

#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
  int pid, pid_terminated, status;
  char cmd[100];

  printf("Command (OR quit)? "); scanf("%s",cmd);

  while (strcmp(cmd,"quit") != 0)
  {
    pid=fork();

    if (pid>0)
    { // COMMENT THE 2 LINES BELOW TO SEE THE ZOMBIES
      pid_terminated = wait(&status);
      printf("PARENT: son %d terminated with exit code %d\n",
        pid_terminated,WEXITSTATUS(status));
    }
    else
    {
      execlp(cmd,cmd,NULL);
      printf("Command not found !!!\n");
      exit(1);
    }

    printf("Command? "); scanf("%s",cmd);
  }

  return 0;
}
```

```c
//========================================================================
// s01.c / JAS
// system()
//------------------------------------------------------------------------

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(void)
{
  int pid;

  printf("before fork\n");
  pid=fork();
  if (pid > 0) {
    wait(NULL);
    printf("I'm the parent (PID=%d)\n\n",getpid()); }
  else {
    printf("I'm the son (PID=%d)\n\n",getpid());
    system("ls /usr/include/s*.h -la"); //NOTE: system() "expands" s*.h
    // try also system("cat s01.c > s01_copy.c");
    printf("\n AFTER system() call\n"); //WHY NOT FAILED, in this case,
like in exec()
    exit(0);
  }
  printf ("PID=%d exiting ...\n",getpid());
  return 0;
}
```

```c
//========================================================================
// OPERATING SYSTEMS / MIEIC / FEUP - Jorge Silva
// Signals - s01.c
// Illustrating some asynchronous  signals
// Kill this process using:
// 1) CTRL-C
// 2) "kill" from another terminal window
//------------------------------------------------------------------------

#include <stdio.h>

int main(void)
{
  printf("I'm going to work very hard !\n");

  for (;;); // Doing some "work" that never ends

  return 0;
}
```

```c
//================================================================
// OPERATING SYSTEMS / MIEIC / FEUP - Jorge Silva
// Signals - s02.c
// Illustrating some asynchronous  signals
// Ignoring SIGINT
// try to kill this process using:
// 1) CTRL-C
// 2) "kill" from another terminal window
//----------------------------------------------------------------

#include <stdio.h>
#include <signal.h>

int main(void)
{
  signal(SIGINT,SIG_IGN);

  printf("I'm going to work very hard !\n");

  for (;;); // Doing some "work" that never ends

  return 0;
}
/*
/usr/users1/dei/jsilva/sope/signals> ./s2
I'm going to work very hard !
^C
^C
^C
Terminated  <------- "kill PID" executed from another terminal
/usr/users1/dei/jsilva/sope/signals>
/usr/users1/dei/jsilva/sope/signals> echo $?
143
/usr/users1/dei/jsilva/sope/signals>
*/
```

```c
//================================================================
// OPERATING SYSTEMS / MIEIC / FEUP - Jorge Silva
// Signals - s03.c
// Illustrating some asynchronous  signals
// Ignoring SIGINT and SIGTERM
// Try to kill this process using:
// 1) CTRL-C
// 2) "kill" from another terminal window
//----------------------------------------------------------------

#include <stdio.h>
#include <signal.h>

int main(void)
{
  signal(SIGINT,SIG_IGN);
  signal(SIGTERM,SIG_IGN);

  printf("I'm going to work very hard !\n");

  for (;;); // Doing some "work" that never ends

  return 0;
}

/*
/usr/users1/dei/jsilva/sope/signals> ./s3
I'm going to work very hard !
Killed   <------------  "ps u" + "kill -KILL PID" from another terminal
/usr/users1/dei/jsilva/sope/signals> echo $?
137
/usr/users1/dei/jsilva/sope/signals>
*/
```

```c
//=================================================================
// OPERATING SYSTEMS / MIEIC / FEUP - Jorge Silva
// Signals - s04.c
// Illustrating some asynchronous  signals
// SIGKILL can't be ignored ...
// try to kill this process using:
// 1) CTRL-C
// 2) "kill" from another terminal window
//-----------------------------------------------------------------

#include <stdio.h>
#include <signal.h>
#include <stdlib.h>

int main(void)
{
  signal(SIGINT,SIG_IGN);
  signal(SIGTERM,SIG_IGN);
  if (signal(SIGKILL,SIG_IGN) == SIG_ERR) // should allways test SIG_ERR ...
  {
    printf("SIGKILL can't be ignored ...!\n");
    exit(1);
  };

  printf("I'm going to work very hard !\n");

  for (;;); // Doing some "work" that never ends

  return 0;
}

/*
/usr/users1/dei/jsilva/sope/signals> ./s04
SIGKILL can't be ignored ...!
/usr/users1/dei/jsilva/sope/signals>
*/
```

```c
//======================================================================
// OPERATING SYSTEMS / MIEIC / FEUP - Jorge Silva
// Signals - s05.c
// SIGKILL can't be caught ...
// try to kill this process using:
// 1) CTRL-C
// 2) "kill" from another terminal window
//----------------------------------------------------------------------

#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

void sigkill_handler(int signo)
{
  printf("SIGKILL received by process %d...!\n", getpid());
}

int main(void)
{
  signal(SIGINT,SIG_IGN);
  signal(SIGTERM,SIG_IGN);
  if (signal(SIGKILL,sigkill_handler) == SIG_ERR)
  {
    printf("SIGKILL can't be caught ...!\n");
    exit(1);
  };

  printf("I'm going to work very hard !\n");

  for (;;); // Doing some "work" that never ends

  return 0;
}
/*
/usr/users1/dei/jsilva/sope/signals> ./s05
SIGKILL can't be caught ...!
/usr/users1/dei/jsilva/sope/signals>
*/
```

```c
//========================================================================
// OPERATING SYSTEMS / MIEIC / FEUP - Jorge Silva
// Signals – s06.c
// Illustrating some asynchronous  signals
// try to kill this process using:
// 1) CTRL-C
// 2) "kill" from another terminal window
//------------------------------------------------------------------------

#include <stdio.h>
#include <signal.h>

void sigint_handler(int signo)
{
  printf("I can't be CTRL-C'ed  :)\n");
}

int main(void)
{
  signal(SIGINT,sigint_handler);

  printf("I'm going to work very hard !\n");

  for (;;); // Doing some "work" that never ends

  return 0;
}
```

```c
//========================================================================
// OPERATING SYSTEMS / MIEIC / FEUP - Jorge Silva
// Signals – s07.c
// Illustrating some asynchronous  signals
// try to kill this process using:
// 1) CTRL-C
// 2) "kill" from another terminal window
//------------------------------------------------------------------------

#include <stdio.h>
#include <signal.h>
#include <unistd.h>

void sigint_handler(int signo)
{
  printf("I can't be CTRL-C'ed  :)\n");
  sleep(5); // <------ NOTE THIS
}

int main(void)
{
  signal(SIGINT,sigint_handler);

  printf("I'm going to work very hard !\n");

  for (;;); // Doing some "work" that never ends

  return 0;
}
```

```c
//=================================================================
// OPERATING SYSTEMS / MIEIC / FEUP - Jorge Silva
// Signals – s08.c
// Illustrating some asynchronous  signals
// Installing a handler for SIGINT & SIGTERM
// How to end this process ?
// - "kill" from another terminal (sends SIGTERM)
// Try with "kill -KILL pid"
//-----------------------------------------------------------------

#include <stdio.h>
#include <signal.h>

void sigint_handler(int signo)
{
  printf("SIGINT received ... \n");
  return;
}

void sigterm_handler(int signo)
{
  printf("SIGTERM received ... \n");
  return;
}

int main(void)
{
  // installing handler for CTRL-C signal (SIGINT)
  signal(SIGINT,sigint_handler);

  // installing handler for default "kill" command action (SIGTERM)
  signal(SIGTERM,sigterm_handler);
  //signal(SIGTERM,SIG_IGN);

  printf("I'm going to work very hard !\n");
  for (;;);  // Doing some "work" that never ends

  return 0;
}
```

```c
//=========================================================================
// OPERATING SYSTEMS / MIEIC / FEUP - Jorge Silva
// Signals - s09.c
// Illustrating some synchronous signals
//-------------------------------------------------------------------------

#include <stdio.h>
#include <signal.h>
#include <stdlib.h>

void sigsegv_handler(int signo)
{
  printf("In SIGSEGV handler\n");
  printf("Error: forbidden memory access !!!\n");
  exit(1); // "return" => infinite cycle
}

int main(void)
{
  int  *year;

  //UNCOMMENT THE 2 ALTERNATIVES
  /*
  if (signal(SIGSEGV,SIG_IGN)==SIG_ERR)
  {
  printf("SIGSEGV can't be ignored ...!\n");
  exit(1); // no error, but it can't be ignored
  }
  */
  signal(SIGSEGV, sigsegv_handler);

  year = (int *) 100;
  *year = 2010;

  printf("ano = %d\n",*year);

  return 0;
}
```

```c
//========================================================================
// OPERATING SYSTEMS / MIEIC / FEUP - Jorge Silva
// Signals - s12.c
// Installing a handler for CTRL-C and returning to default handler
//------------------------------------------------------------------------

#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>

int main(void)
{
  void (*oldhandler)(int);   // <---- NOTE THIS

  printf("I can be CTRL-C'ed\n");
  sleep(5);

  oldhandler = signal(SIGINT, SIG_IGN);
  printf("\nI'm protected from Ctrl-C now \n");
  sleep(5);

  signal(SIGINT, oldhandler);
  printf("\nI'm vulnerable again!\n");
  sleep(5);

  printf("Bye.\n");
  exit(0);
}
```

```c
//=========================================================================
// OPERATING SYSTEMS / MIEIC / FEUP - Jorge Silva
// Signals - s13.c
// Installing a handler for SIGALRM
// An example of a race condition
//-------------------------------------------------------------------------

#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>

int alarmflag = 0;

void alarmhandler(int signo)
{
  printf("Alarm received ...\n");
  alarmflag = 1;
}

int main(void)
{
  signal(SIGALRM, alarmhandler);
  alarm(5);
  printf("Pausing ...\n");
  if (!alarmflag) pause(); // RACE CONDITION ...!
  printf("Ending ...\n");
  exit(0);
}
```

```c
//======================================================================
// OPERATING SYSTEMS / MIEIC / FEUP - Jorge Silva
// Signals - s14.c
// POSIX signals APIs
// Installing a handler for SIGINT

// IMPORTANT NOTE:
// You should always use POSIX APIs.
// signal() call is deprecated
// It was used in the previous exemples
// because it is easier to introduce the signal concepts
//----------------------------------------------------------------------

#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <stdlib.h>

void sigint_handler(int sig) {
  printf("AUUU! Received signal %d\n",sig);
}

int main(void)
{
  struct sigaction action;

  // prepare the 'sigaction struct'
  action.sa_handler = sigint_handler;
  sigemptyset(&action.sa_mask);
  action.sa_flags = 0;

  // install the handler
  sigaction(SIGINT,&action,NULL);

  while(1)
  {
    printf("Hello !\n"); sleep(1);
  }
  exit(0);
}
```

```c
//=================================================================
// OPERATING SYSTEMS / MIEIC / FEUP - Jorge Silva
// Signals - s15.c
// Installing a handler for CTRL-C and returning to default handler
//-----------------------------------------------------------------

#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>

int main(void)
{
  struct sigaction action, orig_action;

  printf("I can be CTRL-C'ed\n");
  sleep(5);

  // prepare the 'sigaction struct' for ignoring SIGINT
  action.sa_handler = SIG_IGN;
  sigemptyset(&action.sa_mask);
  action.sa_flags = 0;

  // ignore SIGINT and get the original handler
  sigaction(SIGINT,&action,&orig_action);

  printf("\nI'm protected from Ctrl-C now \n");
  sleep(5);

  // set the original handler
  sigaction(SIGINT,&orig_action,NULL);

  printf("\nI'm vulnerable again!\n");
  sleep(5);

  printf("Bye.\n");
  exit(0);
}
```

```c
//========================================================================
// OPERATING SYSTEMS / MIEIC / FEUP - Jorge Silva
// Signals - s16.c
// POSIX signals APIs
// Using 'sigsuspend' to solve the race condition problem
// of a previous SIGALRM example
//------------------------------------------------------------------------

#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <stdlib.h>

void alarm_handler(int signo)
{
  printf("Alarm received ...\n");
  //alarmflag = 1; <--- NOT NEEDED WITH 'sigsuspend'
}

int main(void)
{
  struct sigaction action;
  sigset_t sigmask;

  // install SIGALRM handler
  action.sa_handler = alarm_handler;
  sigemptyset(&action.sa_mask); //all signals are delivered
  action.sa_flags = 0;
  sigaction(SIGALRM,&action,NULL);

  // prepare mask for 'sigsuspend'
  sigfillset(&sigmask);         //all signals blocked ...
  sigdelset(&sigmask,SIGALRM);  //...except SIGALRM

  alarm(5);

  printf("Pausing ...\n");
  //while (!alarmflag) pause(); //REPLACED BY 'sigsuspend'
  sigsuspend(&sigmask);

  printf("Ending ...\n");
  exit(0);
}
```

```c
//======================================================================
// OPERATING SYSTEMS / MIEIC / FEUP - Jorge Silva
// Signals - s17.c
// Program that blocks SIGTERM signal for n_seconds, using sigprocmask()
// After that the signal is unblocked and the queued signal is handled
//----------------------------------------------------------------------

#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>

int signal_received = 0;

void sigterm_handler(int signo)
{
  printf("Executing SIGTERM handler\n");
  signal_received = 1;
}

int main (int argc, char * argv[])
{
  sigset_t mask;
  sigset_t orig_mask;
  struct sigaction action;
  int n_seconds = 30;

  if (argc == 2)
    n_seconds = atoi(argv[1]);

  //clean all fields of 'action', including 'sa_mask' and 'sa_flags'
  memset (&action, 0, sizeof(action));
  //install handler for SIGTERM
  action.sa_handler = sigterm_handler;
  sigaction(SIGTERM, &action, 0);

  // temporarily block SIGTERM
  sigemptyset (&mask);
  sigaddset (&mask, SIGTERM);

  //set new mask and get original mask
  sigprocmask(SIG_BLOCK, &mask, &orig_mask);

  printf("Sleeping for %d seconds ...\n",n_seconds);
  sleep (n_seconds);

  // set original signal mask (unblocks SIGTERM)
  sigprocmask(SIG_SETMASK, &orig_mask, NULL);
  printf("Mask for SIGTERM removed\n");

  if (signal_received)
    printf("Signal received\n");

  return 0;
}
```

```c
// OPERATING SYSTEMS
// PIPES- Examples from lectures
// Father process reads 2 integers from the keyboard
// and sends them to its son, through a pipe.
// Son reads the integers from the pipe,
// computes their sum and displays the result.
// JAS
// pipe00.c

#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

#define READ 0
#define WRITE 1

int main(void)
{
  int fd[2];
  pid_t pid;

  pipe(fd);

  pid = fork();

  if (pid >0) //pai
  {
    int a[2];
    printf("PARENT:\n");
    printf("x y ? "); scanf("%d %d",&a[0],&a[1]);
    close(fd[READ]);
    write(fd[WRITE],a,2*sizeof(int));
    close(fd[WRITE]);
  }
  else //filho
  {
    int b[2];
    //printf("SON:\n");
    close(fd[WRITE]);
    read(fd[READ],b,2*sizeof(int));
    printf("SON:\n"); //WHY HERE AND NOT ABOVE ...?!
    printf("x + y = %d\n", b[0]+b[1]);
    close(fd[READ]);
  }
  return 0;
}
```

```c
// SISTEMAS OPERATIVOS
// PIPES- Exemplos das aulas teóricas
// Programa que mostra um ficheiro, página a página,
// usando o paginador do UNIX
// JAS
// pipe01.c

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

#define   MAXLINE   1000
#define   PAGER "/bin/more"

int main(int argc, char *argv[])
{
      char   line[MAXLINE];
      FILE   *fpin, *fpout;

      if (argc != 2) { printf("usage: %s filename\n",argv[0]); exit(1); }
      if ((fpin = fopen(argv[1], "r")) == NULL)
      {
            fprintf(stderr,"can't open %s", argv[1]); exit(1);
      }
      if ((fpout = popen(PAGER, "w")) == NULL)
      {
            fprintf(stderr,"popen error"); exit(1);
      }
      /* copy filename contents to pager - file=argv[1] */
      while (fgets(line, MAXLINE, fpin) != NULL)
      {
            if (fputs(line, fpout) == EOF)
            {
                  printf("fputs error to pipe"); exit(1);
            }
      }
      if (ferror(fpin))
      {
            fprintf(stderr,"fgets error"); exit(1);
      }
      if (pclose(fpout) == -1)
      {
            fprintf(stderr,"pclose error");
            exit(1); }
      exit(0);
}
```

```c
// SISTEMAS OPERATIVOS
// PIPES- Exemplos das aulas teoricas
// Programa que mostra um ficheiro, pagina a pagina,
// usando o paginador do UNIX/LINUX
//
// VERSAO SIMPLIFICADA DE pipe01.c (SEM TESTES DE ERRO)
// JAS
// pipe01_s.c
// EXECUCAO: ./pipe01_s pipe01_s.c

#include <stdio.h>
#include <stdlib.h>
#define MAXLINE 1000
#define PAGER "/bin/more"

int main(int argc, char *argv[])
{
  char line[MAXLINE];
  FILE *fpin, *fpout;

  if (argc != 2) { printf("usage: %s filename\n",argv[0]); exit(1); }

  fpin = fopen(argv[1], "r");
  fpout = popen(PAGER, "w");

  while (fgets(line, MAXLINE, fpin) != NULL)
    fputs(line, fpout) == EOF;

  pclose(fpout);

  exit(0);
}
```

```c
//=====================================================================

// SISTEMAS OPERATIVOS
// PIPES- Exemplos das aulas teóricas
// JAS
// Programa que usa um filtro que converte maiúsculas em minúsculas
// pipe02a.c (a executar em conjunto com pipe02b.c = filtro)

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

#define MAXLINE 1000

int main(void)
{
      char line[MAXLINE];
      FILE *fpin;

      if ((fpin=popen("./pipe02b","r")) == NULL)
      {printf("popen error"); exit(1);}
      for ( ; ; )
      { fputs("prompt > ",stdout); fflush(stdout);
      if (fgets(line,MAXLINE,fpin) == NULL) break;
      if (fputs(line,stdout) == EOF)
      { fprintf(stderr,"fputs error"); exit(1);}
      }
      if (pclose(fpin)==-1)
      {fprintf(stderr,"pclose error"); exit(1);}
      putchar('\n');
      exit(0);
}

//=====================================================================

// SISTEMAS OPERATIVOS
// PIPES - Exemplos das aulas teóricas
// JAS
// Filtro que converte maiúsculas em minúsculas
// pipe02b.c (a executar em conjunto com pipe02a.c)

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

int main(void)
{
      int c;

      while ((c=getchar()) != EOF)
      {
            if (isupper(c)) c=tolower(c);
            if (putchar(c)==EOF)
            {
                  printf("output error"); exit(1);
            }
            if (c=='\n') fflush(stdout);
      }
      exit(0);
}
```

```c
//========================================================================

// SISTEMAS OPERATIVOS
// PIPES - Exemplos das aulas teóricas
// JAS (adaptado de Stevens)
// Programa que usa um coprocesso para fazer contas de somar (!)
// pipe03a.c (a executar em conjunto com pipe03b.c = coprocesso)

#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>

#define MAXLINE 1000
#define READ    0
#define WRITE   1

void sig_pipe(int signo);
void err_sys(char *msg);
void err_msg(char *msg);

int main(void)
{
      int     n, fd1[2], fd2[2];
      pid_t   pid;
      char    line[MAXLINE];

      if (signal(SIGPIPE, sig_pipe)==SIG_ERR)
            err_sys("signal error");
      if (pipe(fd1)<0 || pipe(fd2)<0)
            err_sys("pipe error");
      if ((pid=fork())<0) err_sys("fork error");
      else
            if (pid>0) /* PARENT */
            {
                  close(fd1[READ]); close(fd2[WRITE]);
                  printf("Input 2 numbers (END = CTRL-D): ");
                  while (fgets(line, MAXLINE,stdin) != NULL)
                  {
                        n=strlen(line);
                        if (write(fd1[WRITE],line,n) != n)
                              err_sys("write error to pipe");
                        if ((n=read(fd2[READ],line,MAXLINE)) < 0)
                              err_sys("read error from pipe");
                        if (n==0) {err_msg("child closed pipe"); break;}
                        printf("sum = ");
                        line[n]=0;
                        if (fputs(line,stdout)==EOF) err_sys("fputs error");
                        printf("Input 2 numbers  (END 0 CTRL-D): ");

                  }
                  if (ferror(stdin)) err_sys("fgets error on stdin");
                  exit(0);
            }
```

```
            else        /* CHILD */
            {
                  close(fd1[WRITE]); close(fd2[READ]);
                  if (fd1[READ] != STDIN_FILENO)
                  {
                        if (dup2(fd1[READ],STDIN_FILENO) != STDIN_FILENO)
                              err_sys("dup2 error to stdin");
                        close(fd1[READ]);
                  }
                  if (fd2[WRITE] != STDOUT_FILENO)
                  {
                        if (dup2(fd2[WRITE],STDOUT_FILENO) != STDOUT_FILENO)
                              err_sys("dup2 error to stdout");
                        close(fd2[WRITE]);
                  }
                  if (execl("./pipe03b","pipe03b",NULL) < 0)
                        err_sys("execl error");
            }
            return 0;
}

void sig_pipe(int signo)
{
      printf("SIGPIPE caught\n");
      exit(1);
}

void err_sys(char *msg)
{
      fprintf(stderr,"%s\n",msg);
      exit(1);
}

void err_msg(char *msg)
{
      printf("%s\n",msg); return;
}
```

```c
// SISTEMAS OPERATIVOS
// PIPES - Exemplos das aulas teóricas
// Programa que usa um coprocesso para fazer contas de somar (!)
// pipe03a_s.c (versão simplificada, sem testes de erro, de pipe03a.c)
// (a executar em conjunto com pipe03b.c)
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>

#define MAXLINE 1000
#define READ    0
#define WRITE   1

void sig_pipe(int signo);
void err_sys(char *msg);
void err_msg(char *msg);

int main(void)
{
      int     n, fd1[2], fd2[2];
      pid_t   pid;
      char    line[MAXLINE];

      signal(SIGPIPE, sig_pipe);
      pipe(fd1);
      pipe(fd2);
      pid=fork();
      if (pid>0) // PARENT
      {
            close(fd1[READ]); close(fd2[WRITE]);
            printf("Input 2 numbers (END = CTRL-D): ");
            while (fgets(line, MAXLINE,stdin) != NULL)
            {
                  n=strlen(line);
                  write(fd1[WRITE],line,n); // null ending char is not send !
                  n=read(fd2[READ],line,MAXLINE); //waits for answer (= sum)
                  if (n==0)
                  {
                        err_msg("child closed pipe"); break;
                  }
                  line[n]=0; // null ending char is not received, so "add" it
                  printf("sum = %s",line);
                  printf("Input 2 numbers  (END 0 CTRL-D): ");
            }
            exit(0);
      }
      else       // CHILD
      {
            close(fd1[WRITE]); close(fd2[READ]);
            dup2(fd1[READ],STDIN_FILENO);    // redirect I/O of the coprocess
            dup2(fd2[WRITE],STDOUT_FILENO);  // to the pipes
            if (execl("./pipe03b","pipe03b",NULL) < 0) // execute the coprocess
                  err_sys("execl error");
      }
      return 0;
}
```

```c
void sig_pipe(int signo)
{
      printf("SIGPIPE caught\n");
      exit(1);
}

void err_sys(char *msg)
{
      fprintf(stderr,"%s\n",msg);
      exit(1);
}

void err_msg(char *msg)
{
      printf("%s\n",msg);
      return;
}
```

```c
//========================================================================

// SISTEMAS OPERATIVOS
// PIPES - Exemplos das aulas teóricas
// JAS (adaptado de Stevens)
// Coprocesso para fazer contas de somar (!)
// pipe03b.c (a executar em conjunto com pipe03a.c)

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

#define MAXLINE 100

int main(void)
{
      int    n, int1, int2;
      char   line[MAXLINE];

      while ( (n = read(STDIN_FILENO, line, MAXLINE)) > 0)
      {
            line[n] = 0;  // null terminate
            if (sscanf(line, "%d%d", &int1, &int2) == 2)
            {
                  sprintf(line, "%d\n", int1 + int2);
                  n = strlen(line);
                  if (write(STDOUT_FILENO, line, n) != n)
                  {
                        fprintf(stderr,"write error"); exit(1);
                  }
            }
            else
            {
                  if (write(STDOUT_FILENO, "invalid args\n", 13) != 13)
                  {
                        fprintf(stderr,"write error"); exit(1);
                  }
            }
      }
      exit(0);
}
```

```c
//========================================================================
// SISTEMAS OPERATIVOS
// FIFOS - Exemplos das aulas teóricas
//
// PROGRAMA reader
// fifo01a.c
// deve correr conjuntamente com fifo01b.c
// Experimentar:
// 1) ./fifo01a & ./fifo01b & ./fifo01b & ./fifo01b &
// 2) correr fifo01a numa janela de comando e fifo01b noutra
//    e interpretar resultado;
// lancar em execucao primeiro fifo01a e depois fifo01b e depois o inverso

#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/file.h>

int readline(int fd, char *str);

int main(void)
{
      int   fd;
      char  str[100];

      mkfifo("myfifo",0660);
      fd=open("myfifo",O_RDONLY);

      putchar('\n');
      while(readline(fd,str)) printf("%s",str);
      close(fd);
      return 0;
}


int readline(int fd, char *str)
{
      int n;

      do
      {
            n = read(fd,str,1);
      }
      while (n>0 && *str++ != '\0');
      return (n>0);
}
```

```c
//========================================================================
// SISTEMAS OPERATIVOS
// FIFOS - Exemplos das aulas teóricas
//
// PROGRAMA writer
// fifo01b.c
// deve correr conjuntamente com fifo01a.c
// Experimentar:
// 1) ./fifo01a & ./fifo01b & ./fifo01b & ./fifo01b &
// 2) correr fifo01a numa janela de comando e fifo01b noutra
//    e interpretar resultado;
// lancar em execucao primeiro fifo01a e depois fifo01b e depois o inverso


#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/file.h>

int main(void)
{
      int   fd, messagelen, i;
      char  message[100];

      do
      {
            fd=open("myfifo",O_WRONLY);
            if (fd==-1) sleep(1);
      }
      while (fd==-1);

      for (i=1; i<=3; i++)
      {
            sprintf(message,"Hello no. %d from process no. %d\n", i, getpid());
            messagelen=strlen(message)+1;
            write(fd,message,messagelen);
            sleep(3);
      }
      close(fd);
      return 0;
}
```

```c
/* SISTEMAS OPERATIVOS
   Arquitectura cliente-servidor

   Programa servidor - srv_01.c

   O cliente envia o nome do utilizador ao servidor
   e este escreve no ecrã "Username has arrived".

   Servidor faz leitura do FIFO com espera activa.
*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/file.h>
#include <signal.h>
#include <errno.h>
#include <string.h>

#define MAX_MSG_LEN 20

int main(void)
{
 int   fd,n;
 char  str[MAX_MSG_LEN];

 if (mkfifo("/tmp/requests",0660)<0)
  if (errno==EEXIST) printf("FIFO '/tmp/requests' already exists\n");
  else printf("Can't create FIFO\n");
 else printf("FIFO '/tmp/requests' sucessfully created\n");

 if ((fd=open("/tmp/requests",O_RDONLY)) !=-1)
  printf("FIFO '/tmp/requests' openned in READONLY mode\n");

 do
  {
    n=read(fd,str,MAX_MSG_LEN);               // QUAL É O PROBLEMA DESTA SOLUÇÃO ?
    if (n>0) printf("%s has arrived\n",str);  // O QUE ACONTECE Q.DO NÃO HOUVER CLIENTES ?
    sleep(1);                                 // COMO RESOLVÊ-LO ?
  } while (strcmp(str,"SHUTDOWN")!=0);

  close(fd);
  if (unlink("/tmp/requests")<0)
   printf("Error when destroying FIFO '/tmp/requests'\n");
  else
   printf("FIFO '/tmp/requests' has been destroyed\n");
  exit(0);
}
```

```c
/* SISTEMAS OPERATIVOS
   Arquitectura cliente-servidor

   Programa cliente  - cli_01.c = cli_02.c

   O cliente envia o nome do utilizador ao servidor
   e este escreve no ecrã "Username has arrived"
*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/file.h>
#include <string.h>

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

 if (argc!=2) {
  printf("Usage: cli_01 <username>\n");
  exit(1);
 }

 fd=open("/tmp/requests",O_WRONLY);
 if (fd == -1) {
  printf("Oops !!! Server is closed !!!\n");
  exit(1);
 }

 printf("FIFO 'requests' openned in WRITEONLY mode\n");

 write(fd,argv[1],strlen(argv[1])+1);
 close(fd);
 return 0;
}
```

```c
/* SISTEMAS OPERATIVOS
   Arquitectura cliente-servidor

   Programa servidor - srv_02.c

   O cliente envia o nome do utilizador ao servidor
   e este escreve no ecran "Username has arrived".

   Servidor faz leitura do FIFO sem espera activa.
*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/file.h>
#include <signal.h>
#include <errno.h>
#include <string.h>

#define MAX_MSG_LEN 20

int main(void)
{
 int   fd, n, fd_dummy;
 char  str[MAX_MSG_LEN];

 if (mkfifo("/tmp/requests",0660)<0)
  if (errno==EEXIST) printf("FIFO '/tmp/requests' already exists\n");
  else printf("Can't create FIFO\n");
 else printf("FIFO '/tmp/requests' sucessfully created\n");

 if ((fd=open("/tmp/requests",O_RDONLY)) !=-1)
  printf("FIFO '/tmp/requests' openned in READONLY mode\n");

 // UMA SOLUÇÃO P/PROBLEMA DE srv_01.c (busy waiting)
 // EXISTE OUTRA SOLUÇÃO ?
 if ((fd_dummy=open("/tmp/requests",O_WRONLY)) !=-1)
  printf("FIFO '/tmp/requests' openned in WRITEONLY mode\n");

 do
  {
    n=read(fd,str,MAX_MSG_LEN);
    if (n>0) printf("%s has arrived\n",str);
  } while (strcmp(str,"SHUTDOWN")!=0);

  close(fd);
  close(fd_dummy);
  if (unlink("/tmp/requests")<0)
   printf("Error when destroying FIFO '/tmp/requests'\n");
  else
   printf("FIFO '/tmp/requests' has been destroyed\n");
  exit(0);
}
```

```c
/* SISTEMAS OPERATIVOS
   Arquitectura cliente-servidor

   Programa servidor - srv_03.c

   O cliente envia um código de operação e nome do utilizador ao servidor
   e este escreve no ecrã "<Username> has requested operation <opcode>".
   O servidor termina quando receber 'opcode' igual a zero.
*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/file.h>
#include <signal.h>
#include <errno.h>
#include <string.h>

#define MAX_NAME_LEN 20

int main(void)
{
 int   fd, fd_dummy;
 char  name[MAX_NAME_LEN];
 int opcode;

 if (mkfifo("/tmp/requests",0660)<0)
  if (errno==EEXIST) printf("FIFO '/tmp/requests' already exists\n");
  else printf("Can't create FIFO\n");
 else printf("FIFO '/tmp/requests' sucessfully created\n");

 if ((fd=open("/tmp/requests",O_RDONLY)) !=-1)
  printf("FIFO '/tmp/requests' openned in READONLY mode\n");

 if ((fd_dummy=open("/tmp/requests",O_WRONLY)) !=-1)
  printf("FIFO '/tmp/requests' openned in WRITEONLY mode\n");

 do
  {
    read(fd,&opcode,sizeof(int));
    if (opcode!=0) {
     read(fd,name,MAX_NAME_LEN);
     printf("%s has requested operation %d\n",name,opcode);
    }
  } while (opcode!=0);

  close(fd);
  close(fd_dummy);
  if (unlink("/tmp/requests")<0)
   printf("Error when destroying FIFO '/tmp/requests'\n");
  else
   printf("FIFO '/tmp/requests' has been destroyed\n");
  exit(0);
}
```

```c
/* SISTEMAS OPERATIVOS
   Arquitectura cliente-servidor

   Programa cliente  - cli_03.c

   O cliente envia um código de operação e nome do utilizador ao servidor
   e este escreve no ecrã "<Username> has requested operation <opcode>".
*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/file.h>
#include <string.h>

#define MAX_MSG_LEN 20

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

 if (argc!=2 && argc!=3) {
  printf("Usage: cli_03 <opcode> <username> OR cli_03 0\n");
  exit(1);
 }

 fd=open("/tmp/requests",O_WRONLY);
 if (fd == -1) {
  printf("Oops !!! Service is closed !!!\n");
  exit(1);
 }

 printf("FIFO 'requests' openned in WRITEONLY mode\n");

 // QUAL É O PROBLEMA DESTE CÓDIGO ?
 // (considerar a existência de múltiplos clientes)
 // A FAZER: implementar a solução correcta
 opcode=atoi(argv[1]);
 write(fd,&opcode,sizeof(int));
 if (opcode!=0) {
  write(fd,argv[2],strlen(argv[2])+1);
 }
 close(fd);
 return 0;
}
```

```c
//=========================================================================
// THREADS - examples
// t01.c
// A program that launches 2 threads and waits for them to end
// Illustrating thread execution interleaving
//-------------------------------------------------------------------------

#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

#define NUM_CHARS 10000

void *thr_func(void *arg)
{
  int i;

  fprintf(stderr, "Starting thread %s\n", (char *) arg);
  for (i = 0; i < NUM_CHARS; i++)
    write(STDOUT_FILENO,(char *) arg,1);
  return NULL;
}

int main(void)
{
  pthread_t tid1, tid2;

  printf("Hello from main thread\n");
  pthread_create(&tid1, NULL, thr_func, "A");
  pthread_create(&tid2, NULL, thr_func, "B");
  pthread_join(tid1,NULL);
  pthread_join(tid2,NULL);
  return 0;
}

//=========================================================================
// THREADS - examples
// t02.c
// What may happen if the main thread is the first one to end ... :-(
//-------------------------------------------------------------------------

#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

void *thr_func(void *arg)
{
  sleep(3);
  printf("Hello from auxiliar thread\n");
  return NULL;
}

int main(void)
{
  pthread_t tid;

  printf("Hello from main thread\n");
  pthread_create(&tid, NULL, thr_func, NULL);
  return 0;
}
```

```c
//========================================================================
// THREADS - examples
// t03.c
// - A child thread can continue running after the main thread end !!!
// - Passing info between threads using global variables
//------------------------------------------------------------------------

#include <stdio.h>
#include <pthread.h>

int global;

void *thr_func(void *arg)
{
  printf("Aux thread: %d\n", global);
  return NULL;
}

int main(void)
{
  pthread_t tid;
  global = 20;

  printf("Main thread: %d\n", global);
  pthread_create(&tid, NULL, thr_func, NULL);
  pthread_exit(NULL);
}

//========================================================================
// THREADS - examples
// t04.c
// - Passing info bidirectionally, using global variables
// - Waiting for the end of a thread (alternative: use sync. mechan.)
//------------------------------------------------------------------------

#include <stdio.h>
#include <pthread.h>

int global;

void *thr_func(void *arg)
{
  global = 20;
  printf("Aux thread: %d\n", global);
  return NULL;
}

int main(void)
{
  pthread_t tid;

  global = 10;
  printf("Main thread: %d\n", global);
  pthread_create(&tid, NULL, thr_func, NULL);
  pthread_join(tid, NULL);
  printf("Main thread: %d\n", global);
  return 0;
}
```

```c
//========================================================================
// THREADS - examples
// t05.c
// Passing info through thread arguments and return values
//------------------------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

void *thr_func(void *arg)
{
  void *ret;
  int value;

  value = *(int *) arg;
  printf("Aux thread: %d\n",value);
  value++;
  ret = malloc(sizeof(int));
  *(int *)ret = value;
  return ret;
}

int main(void)
{
  pthread_t tid;
  int k = 10;
  void *r;

  pthread_create(&tid, NULL, thr_func, &k);
  pthread_join(tid, &r);
  printf("Main thread: %d\n", *(int *)r);
  free(r);
  return 0;
}
//========================================================================
// THREADS - examples
// t06.c
// Passing arguments to threads - BE CAREFUL !!!
//------------------------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define NUM_THREADS 10

void *printHello(void *threadId)
{
  printf("Thread %2d: Hello World!\n", *(int*)threadId);
  pthread_exit(NULL);
}

int main()
{
  pthread_t tid[NUM_THREADS];
  int rc, t;
  for(t=1; t<= NUM_THREADS; t++){
    printf("Creating thread %d\n", t);
    rc = pthread_create(&tid[t-1], NULL, printHello, &t);
    if (rc)
    {
      printf("ERROR; return code from pthread_create() is %d\n", rc);
      exit(1);
    }
  }
  pthread_exit(NULL);
}
```

```c
//========================================================================
// THREADS - examples
// t07.c
// Passing arguments to threads
// One solution the the "passing arguments to the threads" problem
// (only possible in some situations ... when?)
//------------------------------------------------------------------------

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define NUM_THREADS 10

void *printHello(void *threadId)
{
  printf("Thread %2d: Hello World!\n", (int)threadId);
  pthread_exit(NULL);
}

int main()
{
  pthread_t tid[NUM_THREADS];
  int rc, t;
  for(t=1; t<= NUM_THREADS; t++){
    printf("Creating thread %d\n", t);
    rc = pthread_create(&tid[t-1], NULL, printHello, (void *)t);
    if (rc)
    {
      printf("ERROR; return code from pthread_create() is %d\n", rc);
      exit(1);
    }
  }
  pthread_exit(NULL);
}
```

```c
//========================================================================
// THREADS - examples
// t08.c
// Passing arguments to threads
// Another solution (?) - see execution example after the code
// to the "passing arguments to the threads" problem
//------------------------------------------------------------------------

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define NUM_THREADS 10

void *printHello(void *threadId)
{
  printf("Thread %2d: Hello World!\n", *(int *) threadId);
  pthread_exit(NULL);
}

int main()
{
  pthread_t tid[NUM_THREADS];
  int rc, t;
  int thrArg[NUM_THREADS];

  for(t=1; t<= NUM_THREADS; t++){
    printf("Creating thread %d\n", t);
    thrArg[t-1] = t;
    rc = pthread_create(&tid[t-1], NULL, printHello, &thrArg[t-1]);
    if (rc)
    {
      printf("ERROR; return code from pthread_create() is %d\n", rc);
      exit(1);
    }
  }
  pthread_exit(NULL);
}
```

```c
//============================================================================
// THREADS - examples
// t09.c
// Passing arguments to threads
// The solution to the "passing arguments to the threads" problem:
// allocate space for the arguments in the heap
//----------------------------------------------------------------------------

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define NUM_THREADS 10

void *printHello(void *threadId)
{
  printf("Thread %2d: Hello World!\n", *(int *) threadId);
  free(threadId);
  pthread_exit(NULL);
}

int main()
{
  pthread_t tid[NUM_THREADS];
  int rc, t;
  int *thrArg;

  for(t=1; t<= NUM_THREADS; t++){
    printf("Creating thread %d\n", t);
    thrArg = (int *) malloc(sizeof(t));
    *thrArg = t;
    rc = pthread_create(&tid[t-1], NULL, printHello, thrArg);
    if (rc)
    {
      printf("ERROR; return code from pthread_create() is %d\n", rc);
      exit(1);
    }
  }
  pthread_exit(NULL);
}
//TO DO: modify in order to free the memory allocated in the heap
```

```c
//========================================================================
// THREADS - examples
// t10.c
// What is the danger of using the 'global' variable?
//------------------------------------------------------------------------

#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

#define NUM_ITER 20

int global = 0;

void *thrFunc(void *arg)
{
  while (global++ < NUM_ITER)
  {
    printf("t%d - %d\n",*(int *)arg,global);
    sleep(1); //  <----- COMMENT AND RE-EXECUTE
  }
  return NULL;
}

int main(void)
{
  pthread_t tid1, tid2;
  int t1=1, t2=2; //thread number

  printf("Hello from main thread\n");
  pthread_create(&tid1, NULL, thrFunc, (void *)&t1);
  pthread_create(&tid2, NULL, thrFunc, (void *)&t2);
  pthread_join(tid1,NULL);
  pthread_join(tid2,NULL);
  return 0;
}
```

```c
//========================================================================
// THREADS - examples
// t10a.c
// What is the danger of using the 'global' variable?
//------------------------------------------------------------------------

#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

#define NUM_ITER 20

int global=0;

void *thrFunc(void *arg)
{
  while (global++ < NUM_ITER)
  {
    printf("t%d - %d\n",*(int *)arg,global);
    //sleep(1); //  <----- COMMENT AND RE-EXECUTE
  }
  return NULL;
}

int main(void)
{
  pthread_t tid1, tid2;
  int t1=1, t2=2; //thread number

  printf("Hello from main thread\n");
  pthread_create(&tid1, NULL, thrFunc, (void *)&t1);
  pthread_create(&tid2, NULL, thrFunc, (void *)&t2);
  pthread_join(tid1,NULL);
  pthread_join(tid2,NULL);
  return 0;
}
```

```c
// PRODUCER-CONSUMER PROBLEM
// PRODUCER AND CONSUMER ARE THREADS OF A SINGLE PROCESS
// Illustrates the use of POSIX semaphores for synchronization
// Note: buffer capacity is 1 !!!
// JAS

// QUESTION: why is the mutex,
// usually used in the classical producer-consumer problem
// not needed, in this case?

// NOTE: error return codes are not checked ...
// You must add them.

// prod_cons_1.c
// compilation: gcc prod_cons_1.c -lpthread -lrt -Wall -o prod_cons_1

//========================================================================
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>

//========================================================================
#define NOT_SHARED 0  // sem. is not shared w/other processes

//========================================================================
sem_t empty, full;       // the global semaphores
int data;                // shared buffer; capacity=1 !!!
int numItems;            // number of items to be produced/consumed

//========================================================================
// Put items (1, ..., numItems) into the data buffer and sum them
void *producer(void *arg) {
  int total=0, produced;

  printf("Producer running\n");

  for (produced = 1; produced <= numItems; produced++)
  {
    sem_wait(&empty);
    data = produced;
    total = total+data;
    sem_post(&full);
  }

  printf("Producer: total produced is %d\n",total);

  return NULL;
}
```

```c
//=========================================================================
// Get values from the data buffer and sum them
void *consumer(void *arg) {
  int total = 0, consumed;

  printf("Consumer running\n");

  for (consumed = 1; consumed <= numItems; consumed++)
  {
    sem_wait(&full);
    total = total+data;
    sem_post(&empty);
  }

  printf("Consumer: total consumed is %d\n",total);

  return NULL;
}

//=========================================================================
int main(int argc, char *argv[]) {
  pthread_t pid, cid;

  if (argc != 2)
  {
    fprintf(stderr,"USAGE: %s numItems\n",argv[0]);
    exit(1);
  }

  numItems = atoi(argv[1]); // num. of items to be produced/consumed

  sem_init(&empty, NOT_SHARED, 1);  // sem. empty = 1
  sem_init(&full, NOT_SHARED, 0);   // sem.  full = 0

  printf("Main started.\n");

  pthread_create(&pid, NULL, producer, NULL);
  pthread_create(&cid, NULL, consumer, NULL);

  pthread_join(pid, NULL);
  pthread_join(cid, NULL);

  sem_destroy(&empty);
  sem_destroy(&full);

  printf("Main done.\n");
  return 0;
}
```

```c
// POSIX shared memory & semaphore - usage example
// Program that writes a digit sequence in shared memory and
// waits for a reader (reader.c) to read it
// The reader must write an '*'
// at the beginning of the shared memory region
// for signaling the writer that the shared memory region can be removed
// (another semaphore could have been used instead - TO DO BY STUDENTS)
// JAS

// writer.c
// gcc writer.c -lrt -Wall -o writer (don't forget '-lrt')

//=========================================================================
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h> // For O_* constants
#include <semaphore.h>
#include <sys/mman.h>
#include <sys/mman.h>
#include <sys/types.h>

#define SHM_SIZE 10

//=========================================================================
//names should begin with '/'
char SEM_NAME[] = "/sem1";
char SHM_NAME[] = "/shm1";

//=========================================================================
int main()
{
  int shmfd;
  char *shm, *s;
  sem_t *sem;
  int i, n;
  int sum = 0;

  //create the shared memory region
  shmfd = shm_open(SHM_NAME,O_CREAT|O_RDWR,0600);

  //TO DO BY STUDENTS:
  //try the following alternative for shm_open() call - note the use of
O_EXCL
  // shmfd = shm_open(SHM_NAME,O_CREAT|O_EXCL|O_RDWR,0600);
  //and comment the
  //  if (shm_unlink(SHM_NAME) < 0) ... call at the end of this program
  //Then run this program twice and explain what happens

  if(shmfd<0)
  {
    perror("WRITER failure in shm_open()");
    exit(1);
  }
  if (ftruncate(shmfd,SHM_SIZE) < 0)
  {
    perror("WRITER failure in ftruncate()");
    exit(2);
  }
```

```c
  //attach this region to virtual memory
  shm = (char *) mmap(0,SHM_SIZE,PROT_READ|PROT_WRITE,MAP_SHARED,shmfd,0);
  if(shm == MAP_FAILED)
  {
    perror("WRITER failure in mmap()");
    exit(3);
  }

  //create & initialize semaphore
  sem = sem_open(SEM_NAME,O_CREAT,0600,0);
  if(sem == SEM_FAILED)
  {
    perror("WRITER failure in sem_open()");
    exit(4);
  }

  //write into shared memory region
  s = shm;
  for(i=0; i<SHM_SIZE-1; i++)
  {
    n = i % 10; sum = sum + n;
    *s++ = (char) ('0' + n);
  }
  *s = (char) 0;

  printf("sum = %d\n", sum);

  sem_post(sem);

  //this loop could be replaced by semaphore use
  //TO DO by students
  printf("Busy waiting for 'reader' to read shared memory ...\n");
  while(*shm != '*')
  {
    sleep(1);
  }

  //close and remove shared memory region and semaphore
  sem_close(sem);
  sem_unlink(SEM_NAME);

  if (munmap(shm,SHM_SIZE) < 0)
  {
    perror("WRITER failure in munmap()");
    exit(5);
  }
  if (shm_unlink(SHM_NAME) < 0)
  {
    perror("WRITER failure in shm_unlink()");
    exit(6);
  }

  exit(0);
}
```

```c
// POSIX shared memory & semaphore - usage example
// Program that reads a digit sequence
// written in shared memory by a writer (writer.c)
// After reading, this program writes an '*'
// at the beginning of the shared memory region
// signaling to the writer that  the region can be removed
// (another semaphore could have been used instead - TO DO BY STUDENTS)
// JAS

// reader.c
// gcc reader.c -lrt -Wall -o reader (don't forget '-lrt')

//=========================================================================
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h> // For O_* constants
#include <semaphore.h>
#include <sys/mman.h>
#include <sys/mman.h>
#include <sys/types.h>

#define SHM_SIZE 10

//=========================================================================
//names should begin with '/'
char SEM_NAME[] = "/sem1";
char SHM_NAME[] = "/shm1";

//=========================================================================
int main()
{
  int shmfd;
  char *shm, *s, ch;
  sem_t *sem;
  int sum = 0;

  //open the shared memory region
  shmfd = shm_open(SHM_NAME,O_RDWR,0600);
  if(shmfd<0)
  {
    perror("READER failure in shm_open()");
    exit(1);
  }

  //attach this region to virtual memory
  shm = (char *) mmap(0,SHM_SIZE,PROT_READ|PROT_WRITE,MAP_SHARED,shmfd,0);
  if(shm == MAP_FAILED)
  {
    perror("READER failure in mmap()");
    exit(2);
  }

  //open existing semaphore
  sem = sem_open(SEM_NAME,0,0600,0);
  if(sem == SEM_FAILED)
  {
    perror("READER failure in sem_open()");
    exit(3);
  }
```

```
  //wait for writer to stop writing
  sem_wait(sem);

  //read the message
  s = shm;
  for (s=shm; *s!=0; s++)
  {
    ch = *s;
    putchar(ch);
    sum = sum + (ch - '0');
  }
  printf("\nsum = %d\n", sum);

  //once done signal exiting of reader
  //could be replaced by semaphore use (TO DO by students)
  *shm = '*';

  //close semaphore and unmap shared memory region
  sem_close(sem);

  if (munmap(shm,SHM_SIZE) < 0)
  {
    perror("READER failure in munmap()");
    exit(4);
  }

  exit(0);
}
```

```c
// PRODUCER-CONSUMER PROBLEM
// PRODUCERS AND CONSUMERS ARE THREADS OF A SINGLE PROCESS
// SYNCHRONIZATION USING CONDITION VARIABLES
// JAS

//========================================================================
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

//========================================================================
#define BUFSIZE 8
#define NUMITEMS 100

//========================================================================
int buffer[BUFSIZE];
int bufin = 0;
int bufout = 0;
int items = 0;
int slots = 0;
int sum = 0;

//========================================================================
pthread_mutex_t buffer_lock = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t  slots_cond = PTHREAD_COND_INITIALIZER;
pthread_cond_t  items_cond = PTHREAD_COND_INITIALIZER;
pthread_mutex_t slots_lock = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t items_lock = PTHREAD_MUTEX_INITIALIZER;

//========================================================================
void put_item(int item)
{
  pthread_mutex_lock(&buffer_lock);
  buffer[bufin] = item;
  bufin = (bufin + 1) % BUFSIZE;
  pthread_mutex_unlock(&buffer_lock);
  return;
}

//========================================================================
void get_item(int *itemp)
{
  pthread_mutex_lock(&buffer_lock);
  *itemp = buffer[bufout];
  bufout = (bufout + 1) % BUFSIZE;
  pthread_mutex_unlock(&buffer_lock);
  return;
}

//========================================================================
void *producer(void * arg)
{
  int i;
  for (i = 1; i <= NUMITEMS; i++)
  {
    /* acquire right to a slot */
    pthread_mutex_lock(&slots_lock);
    printf("Producer: available slots = %d\n",slots);
    while (!(slots > 0))
      pthread_cond_wait (&slots_cond, &slots_lock);
```

```
    slots--;
    pthread_mutex_unlock(&slots_lock);
    put_item(i);
    printf("Producer: produced item %3d\n",i);
    /* release right to an item */
    pthread_mutex_lock(&items_lock);
    items++;
    pthread_cond_signal(&items_cond);
    pthread_mutex_unlock(&items_lock);
  }
  pthread_exit(NULL);
}

//========================================================================
void *consumer(void *arg)
{
  int myitem;
  int i;
  for (i = 1; i <= NUMITEMS; i++)
  {
    pthread_mutex_lock(&items_lock);
    printf("Consumer: available items = %d\n",items);
    while(!(items > 0))
      pthread_cond_wait(&items_cond, &items_lock);
    items--;
    pthread_mutex_unlock(&items_lock);
    get_item(&myitem);
    printf("Consumer: consumed item %3d\n",myitem);
    sum += myitem;
    pthread_mutex_lock(&slots_lock);
    slots++;
    pthread_cond_signal(&slots_cond);
    pthread_mutex_unlock(&slots_lock);
  }
  pthread_exit(NULL);
}

//========================================================================
int main(void)
{
  pthread_t prodtid, constid;
  int i, total;
  slots = BUFSIZE;
  total = 0;

  for (i = 1; i <=NUMITEMS; i++)
    total += i;
  printf("The checksum is %d\n", total);

  if (pthread_create(&constid, NULL, consumer, NULL))
  {
    perror("Could not create consumer");
    exit(EXIT_FAILURE);
  }

  if (pthread_create(&prodtid, NULL, producer, NULL))
  {
    perror("Could not create producer");
    exit(EXIT_FAILURE);
  }
```

```
  pthread_join(prodtid, NULL);
  pthread_join(constid, NULL);

  printf("The threads produced the sum %d\n", sum);
  exit(EXIT_SUCCESS);//EXIT_SUCCESS e EXIT_FAILURE <- stdlib.h
}
```

```c
// PRODUCER-CONSUMER PROBLEM
// PRODUCERS AND CONSUMERS ARE INDEPENDENT PROCESSES
// BUFFER IS IN SHARED MEMORY (EXTERNAL TO THE PROCESSES)
// SYNCHRONIZATION USING CONDITION VARIABLES (IN SHARED MEMORY)
// JAS

// PRODUCER program
// prod_01.c (to be run together with cons_01.c)

//========================================================================
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <fcntl.h> // For O_* constants
#include <semaphore.h>
#include <sys/mman.h>
#include <sys/mman.h>
#include <sys/types.h>

//========================================================================
#define SHM_NAME "/shm1"
#define BUFSIZE 5
#define NUMITEMS 50

//========================================================================
typedef struct {
  pthread_mutex_t buffer_lock;
  pthread_cond_t  slots_cond;
  pthread_cond_t  items_cond;
  pthread_mutex_t slots_lock;
  pthread_mutex_t items_lock;
  int buffer[BUFSIZE];
  int bufin;
  int bufout;
  int items;
  int slots;
  int sum;
} Shared_memory;

//========================================================================
Shared_memory * create_shared_memory(char* shm_name, int shm_size)
{
  int shmfd;
  Shared_memory *shm;

  //create the shared memory region
  shmfd = shm_open(SHM_NAME,O_CREAT|O_RDWR,0660); // try with O_EXCL

  if(shmfd<0)
  {
    perror("Failure in shm_open()");
    return NULL;
  }

  //specify the size of the shared memory region
  if (ftruncate(shmfd,shm_size) < 0)
  {
    perror("Failure in ftruncate()");
    return NULL;
  }
```

```
  //attach this region to virtual memory
  shm = mmap(0,shm_size,PROT_READ|PROT_WRITE,MAP_SHARED,shmfd,0);
  if(shm == MAP_FAILED)
  {
    perror("Failure in mmap()");
    return NULL;
  }

  //initialize data in shared memory
  shm->bufin = 0;
  shm->bufout = 0;
  shm->items = 0;
  shm->slots = BUFSIZE;
  shm->sum = 0;

  return (Shared_memory *) shm;
}

//========================================================================
void destroy_shared_memory(Shared_memory *shm, int shm_size)
{
  if (munmap(shm,shm_size) < 0)
  {
    perror("Failure in munmap()");
    exit(EXIT_FAILURE);
  }
  if (shm_unlink(SHM_NAME) < 0)
  {
    perror("Failure in shm_unlink()");
    exit(EXIT_FAILURE);
  }
}

//========================================================================
void init_sync_objects_in_shared_memory(Shared_memory *shm)
{
  pthread_mutexattr_t mattr;
  pthread_mutexattr_init(&mattr);
  pthread_mutexattr_setpshared(&mattr, PTHREAD_PROCESS_SHARED);

  pthread_mutex_init(&shm->buffer_lock, &mattr);
  pthread_mutex_init(&shm->slots_lock, &mattr);
  pthread_mutex_init(&shm->items_lock, &mattr);

  pthread_condattr_t cattr;
  pthread_condattr_init(&cattr);
  pthread_condattr_setpshared(&cattr, PTHREAD_PROCESS_SHARED);

  pthread_cond_init(&shm->slots_cond, &cattr);
  pthread_cond_init(&shm->items_cond, &cattr);
}

//========================================================================
void put_item(int item, Shared_memory *shm)
{
  pthread_mutex_lock(&shm->buffer_lock);
  shm->buffer[shm->bufin] = item;
  shm->bufin = (shm->bufin + 1) % BUFSIZE;
  pthread_mutex_unlock(&shm->buffer_lock);
  return;
}
```

```
//=========================================================================
void *producer(void * arg)
{
  int i;
  Shared_memory *shm = arg;

  printf("In producer thread\n");

  for (i = 1; i <= NUMITEMS; i++)
  {
    // wait for a slot to be available
    pthread_mutex_lock(&shm->slots_lock);
    printf("Producer: available slots = %d\n",shm->slots);
    while (!(shm->slots > 0))
      pthread_cond_wait (&shm->slots_cond, &shm->slots_lock);
    shm->slots--;
    pthread_mutex_unlock(&shm->slots_lock);

    // produce item
    put_item(i,shm);
    printf("Producer: produced item %3d\n",i);

    // update num. produced items and notify consumer
    pthread_mutex_lock(&shm->items_lock);
    shm->items++;
    pthread_cond_signal(&shm->items_cond);
    pthread_mutex_unlock(&shm->items_lock);
  }
  pthread_exit(NULL);
}

//=========================================================================
int main(void){
  pthread_t prodtid;
  int i, total;
  Shared_memory *shmem;

  printf("\nPRODUCER: starting after 5 seconds ...\n");
  sleep(5);

  if ((shmem = create_shared_memory(SHM_NAME, sizeof(Shared_memory))) ==
NULL)
  {
    perror("PRODUCER: could not create shared memory");
    exit(EXIT_FAILURE);
  }

  init_sync_objects_in_shared_memory(shmem);

  // NOT NECESSARILY A THREAD ... COULD BE JUST A CALL TO producer()
FUNCTION
  if (pthread_create(&prodtid, NULL, producer, shmem))
  {
    exit(EXIT_FAILURE);
  }

  pthread_join(prodtid, NULL);

  destroy_shared_memory(shmem, sizeof(Shared_memory));
```

```
  total = 0;
  for (i = 1; i <=NUMITEMS; i++)
    total += i;
  printf("PRODUCER: the checksum is %d\n", total);

  exit(EXIT_SUCCESS);
}
```

```c
// PRODUCER-CONSUMER PROBLEM
// PRODUCERS AND CONSUMERS ARE INDEPENDENT PROCESSES
// BUFFER IS IN SHARED MEMORY (EXTERNAL TO THE PROCESSES)
// SYNCHRONIZATION USING CONDITION VARIABLES (IN SHARED MEMORY)
// JAS

// CONSUMER program
// cons_01.c (to be run together with prod_01.c)

//=========================================================================
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <fcntl.h> // For O_* constants
#include <semaphore.h>
#include <sys/mman.h>
#include <sys/mman.h>
#include <sys/types.h>

//=========================================================================
#define SHM_NAME "/shm1"
#define BUFSIZE 5
#define NUMITEMS 50

//=========================================================================
typedef struct {
  pthread_mutex_t buffer_lock;
  pthread_cond_t  slots_cond;
  pthread_cond_t  items_cond;
  pthread_mutex_t slots_lock;
  pthread_mutex_t items_lock;
  int buffer[BUFSIZE];
  int bufin;
  int bufout;
  int items;
  int slots;
  int sum;
} Shared_memory;

//=========================================================================
Shared_memory * attach_shared_memory(char* shm_name, int shm_size)
{
  // PÔR PROD. E CONSUM. A TENTAR CRIAR E SE NÃO CONSEGUIR FAZ ATTACH
...???
  int shmfd;
  Shared_memory *shm;

  shmfd = shm_open(SHM_NAME,O_RDWR,0660);

  if(shmfd<0)
  {
    perror("Failure in shm_open()");
    return NULL;
  }

  //attach this region to virtual memory
  shm = mmap(0,shm_size,PROT_READ|PROT_WRITE,MAP_SHARED,shmfd,0);
  if(shm == MAP_FAILED)
  {
    perror("Failure in mmap()");
```

```
    return NULL;
  }

  return (Shared_memory *) shm;

}

//========================================================================
void get_item(int *itemp, Shared_memory *shm)
{
  pthread_mutex_lock(&shm->buffer_lock);
  *itemp = shm->buffer[shm->bufout];
  shm->bufout = (shm->bufout + 1) % BUFSIZE;
  pthread_mutex_unlock(&shm->buffer_lock);
  return;
}

//========================================================================
void *consumer(void *arg)
{
  int myitem;
  int i;
  Shared_memory *shm = arg;

  printf("In consumer thread\n");

  for (i = 1; i <= NUMITEMS; i++)
  {
    // wait for an item to be available
    pthread_mutex_lock(&shm->items_lock);
    printf("Consumer: available items = %d\n",shm->items);
    while(!(shm->items > 0))
      pthread_cond_wait(&shm->items_cond, &shm->items_lock);
    shm->items--;
    pthread_mutex_unlock(&shm->items_lock);

    // consume an item
    get_item(&myitem,shm);
    printf("Consumer: consumed item %3d\n",myitem);

    shm->sum += myitem;

    //update num. available slots and notify producer
    pthread_mutex_lock(&shm->slots_lock);
    shm->slots++;
    pthread_cond_signal(&shm->slots_cond);
    pthread_mutex_unlock(&shm->slots_lock);
  }
  pthread_exit(NULL);
}

//========================================================================
int main(void)
{
  pthread_t constid;
  Shared_memory *shmem;

  printf("\nCONSUMER: starting after 10 seconds ...\n");
  sleep(10);
```

```
  if ((shmem = attach_shared_memory(SHM_NAME, sizeof(Shared_memory))) ==
NULL)
  {
    perror("CONSUMER: could not attach shared memory");
    exit(EXIT_FAILURE);
  }

  // NOT NECESSARILY A THREAD ... COULD BE JUST A CALL TO consumer()
FUNCTION
  if (pthread_create(&constid, NULL, consumer, shmem))
  {
    perror("CONSUMER: could not create consumer");
    exit(EXIT_FAILURE);
  }

  pthread_join(constid, NULL);
  printf("CONSUMER: the threads produced the sum %d\n", shmem->sum);

  if (munmap(shmem,sizeof(Shared_memory)) < 0)
  {
    perror("Failure in munmap()");
    exit(EXIT_FAILURE);
  }

  exit(EXIT_SUCCESS);
}
```

```c
// POSIX CONDITION VARIABLES
// Illustration of pthread_cond_broadcast()
// cond_broadc_01.c
// JAS

//========================================================================
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <pthread.h>

//========================================================================
#define NTHREADS    5

//========================================================================
int conditionMet = 0;
pthread_cond_t cond  = PTHREAD_COND_INITIALIZER;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

//========================================================================
// Function to check the return code
// and exit the program if the function call failed
void checkResult(char *string, int err)
{
  if (err != 0)
  {
    printf("Error %d on %s\n", err, string);
    exit(EXIT_FAILURE);
  }
  return;
}

//========================================================================
void *threadFunc(void *arg)
{
  int res;
  int threadNum = *(int *)arg;

  res = pthread_mutex_lock(&mutex);
  checkResult("pthread_mutex_lock()\n", res);

  while (!conditionMet)
  {
    printf("Thread %d blocked because condition is not met\n", threadNum);
    res = pthread_cond_wait(&cond, &mutex);
    checkResult("pthread_cond_wait()\n", res);
  }

  printf("Thread %d executing critical section for 5 seconds ...\n",
threadNum);
  sleep(5);

  res = pthread_mutex_unlock(&mutex);
  checkResult("pthread_mutex_lock()\n", res);
  return NULL;
}
```

```
//============================================================================
int main(int argc, char *argv[])
{
  int res=0;
  int i;
  int threadnum[NTHREADS];
  pthread_t threadId[NTHREADS];

  printf("Main thread: creating %d threads\n", NTHREADS);
  for(i=0; i<NTHREADS; ++i)
  {
    threadnum[i]=i+1;
    res = pthread_create(&threadId[i], NULL, threadFunc, (void*)
&threadnum[i]);
    checkResult("pthread_create()\n", res);
  }
  printf("Main thread: doing some work until condition is met ...\n");
  sleep(10);
  //The condition has occured ...! Don't ask me what condition or why ...

  //Set the flag and wake up any waiting threads
  res = pthread_mutex_lock(&mutex);
  checkResult("pthread_mutex_lock()\n", res);
  conditionMet = 1;
  printf("Main thread: the condition was met;\n waking up all waiting
threads, using pthread_cond_broadcast()...\n");
  res = pthread_cond_broadcast(&cond);
  checkResult("pthread_cond_broadcast()\n", res);

  res = pthread_mutex_unlock(&mutex);
  checkResult("pthread_mutex_unlock()\n", res);

  printf("Main thread: waiting for threads and cleanup\n");
  for (i=0; i<NTHREADS; ++i)
  {
    res = pthread_join(threadId[i], NULL);
    checkResult("pthread_join()\n", res);
  }

  res = pthread_cond_destroy(&cond);
  checkResult("pthread_cond_destroy()\n", res);
  res = pthread_mutex_destroy(&mutex);
  checkResult("pthread_mutex_destroy()\n", res);

  printf("Main thread: completed.\n");
  return 0;
}
```