```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;
}
```