Cálculo de distancia entre aviones que se mueven en el espacio aéreo

En un espacio aéreo de 3 dimensiones (*x*,*y*,*z*) como el que se muestra en la Figura 1 hay mil aviones en movimiento. Para evitar colisiones entre los aviones, estos deben estar a una distancia mínima que está en función de su velocidad, si un avión vuela casi al nivel del mar su velocidad es 300 Km/h, si vuela a 42,000 pies vuela a 900 Km/h. Los aviones inician en una posición aleatoria y conforme transcurre el tiempo (cada iteración) estos se mueven en el espacio de acuerdo a una dirección.

Si un avión *a* detecta que está a una distancia menor que la mínima de otro avión *b*, entonces se debe enviarles un mensaje de advertencia a ambos aviones.

El problema de los aviones es un problema de cómputo intensivo ya que requiere en cada iteración calcular la distancia de cada avión con respecto a todos los demás que están en el espacio, de manera que si son 1000 aviones, cada iteración requiere calcular $(1000^2+1000)/2$ distancias basándonos en las posiciones *x*,*y*,*z* de cada avión.

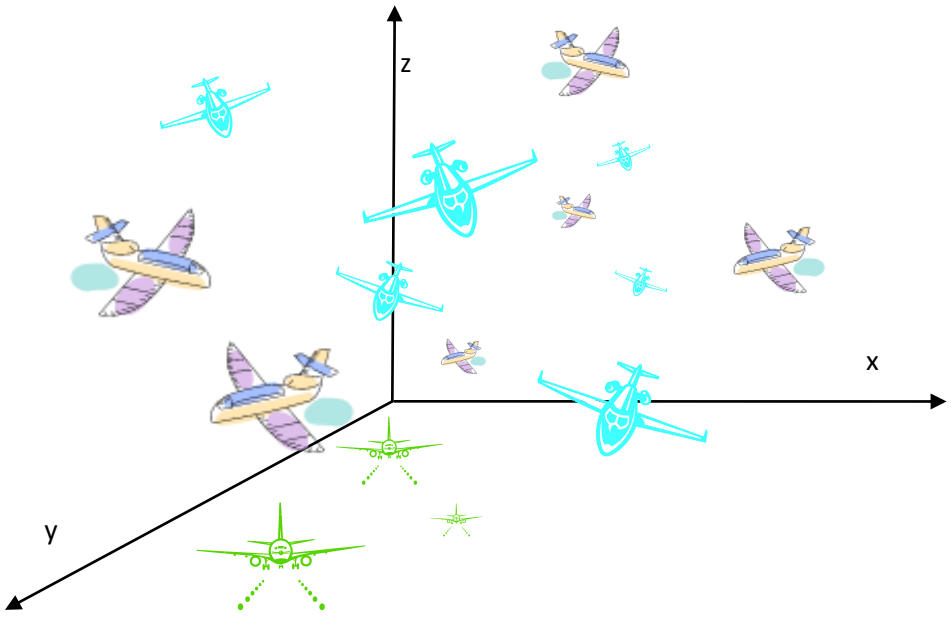


Figura 1. Aviones en el espacio aéreo

Versión serial

El Ejemplo 1 es una versión serial donde cada 10 iteraciones se muestra en pantalla el total de advertencias acumuladas.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#define PI 3.14159

#define SIN(a) sin(a*PI/180.0)
#define COS(a) cos(a*PI/180.0)

#define MAX 1000
#define MINSPEED 300.0  // Km/h
#define MAXSPEED 900.0  // Km/h
#define MAXALT 13000.0  // Metros
#define MINDIST 4000.0 // Distancia mínima en metros entre aviones si estos están

struct AVION
{
	double pos_x;
	double pos_y;
	double pos_z;
	double currspeed;
	double dir;
	double incl;
	int warnings;
};

struct AVION avion[MAX];

double dist(struct AVION a,struct AVION b)
{
	double dist_x,dist_y,dist_z,dist;

	dist_x=a.pos_x-b.pos_x;
	dist_y=a.pos_y-b.pos_y;
	dist_z=a.pos_z-b.pos_z;

	dist=sqrt(dist_x*dist_x+dist_y*dist_y+dist_z*dist_z);
	return(dist);
}

double distmin(struct AVION a,struct AVION b)
{
	double speed;
	double distmin;
	if(a.currspeed>b.currspeed)
		speed=a.currspeed/3.6;
	else
		speed=b.currspeed/3.6;
```

```
	distmin=MINDIST+speed*speed*1.6;
	return(distmin);
}



void inicializa_aviones(struct AVION *x)
{
	int i;
	for(i=0;i<MAX;i++)
	{
		x[i].pos_x=1000*(5000.0 - (double) (rand()%10000));
		x[i].pos_y=1000*(5000.0 - (double) (rand()%10000));
		x[i].pos_z=(double) (rand()%13000);
		x[i].currspeed=MINSPEED+x[i].pos_z/21;
		x[i].dir=(double) (rand()%360);
	}
	return;
}

// Se actualiza cada 1/1000 segundos
void actualiza_avion(struct AVION *a)
{
	if(a->pos_z<13000.0)
		a->pos_z+=0.001;

	a->currspeed=MINSPEED+a->pos_z/21;

	a->pos_x+=0.001*(a->currspeed/3.6)*SIN(a->dir);
	a->pos_y+=0.001*(a->currspeed/3.6)*COS(a->dir);

	return;
}



int main()
{
	int n;
	int i,j;
	int totwarnings=0;
	clock_t t_inicial,t_final;

	inicializa_aviones(avion);

	t_inicial=clock();
	for(n=0;n<2000;n++)
	{
		for(i=0;i<MAX-1;i++)
		{
			for(j=i;j<MAX;j++)
				if(i!=j)

	if(dist(avion[i],avion[j])<distmin(avion[i],avion[j]))
				{
					avion[i].warnings++;
					avion[j].warnings++;
					totwarnings++;
				}
		}
		for(i=0;i<MAX;i++)
```

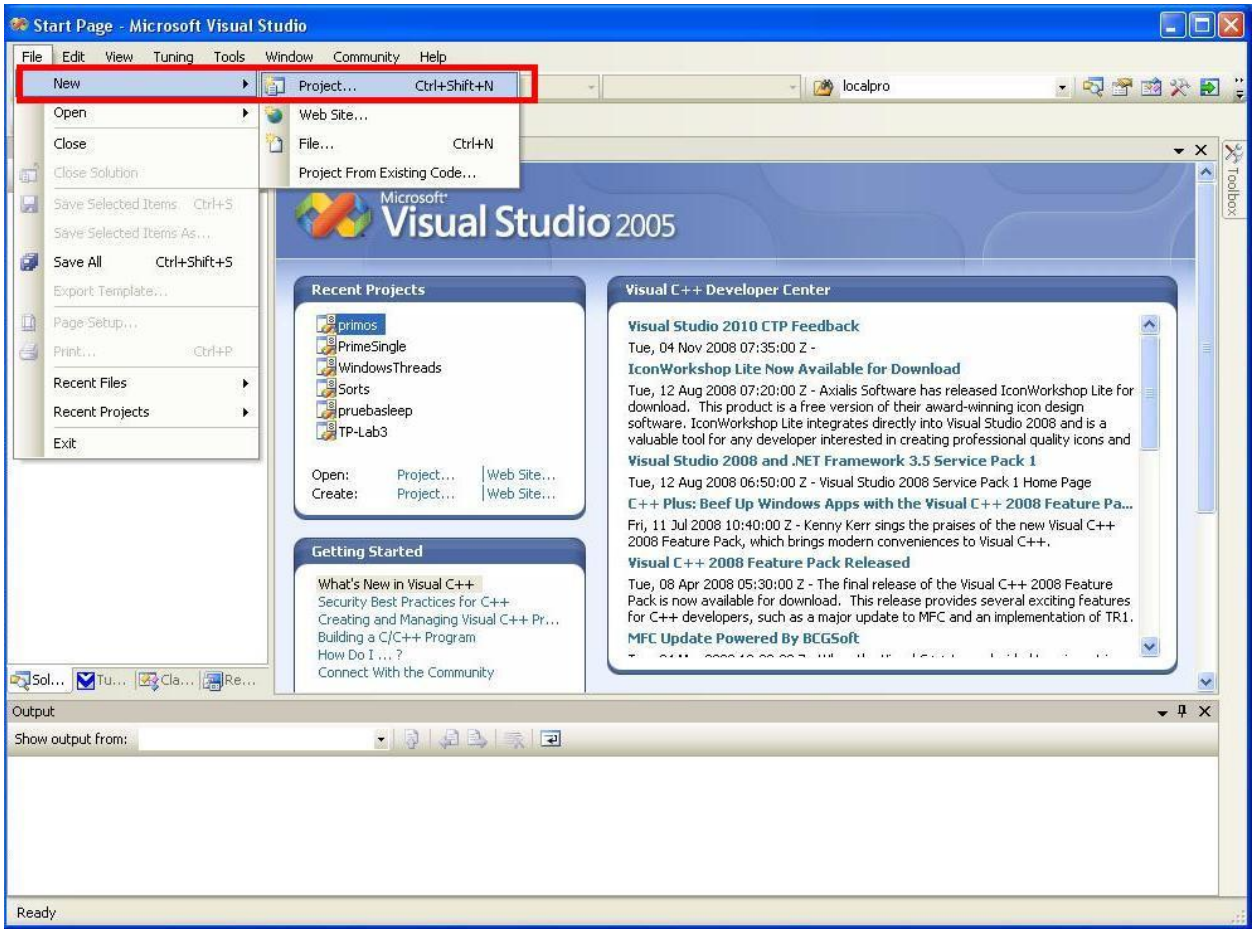
```
                    actualiza_avion(&avion[i]);

             if(n%100==0)
                    printf("Total Warnings = %d\n",totwarnings);
      }
      t_final=clock();
      printf("En %3.6f segundos\n",((float) t_final- (float)t_inicial)/
CLOCKS_PER_SEC);


}
```

Ejemplo 1. Versión serial programa que busca aviones a una distancia menor que la mínima en un espacio de 3 dimensiones.

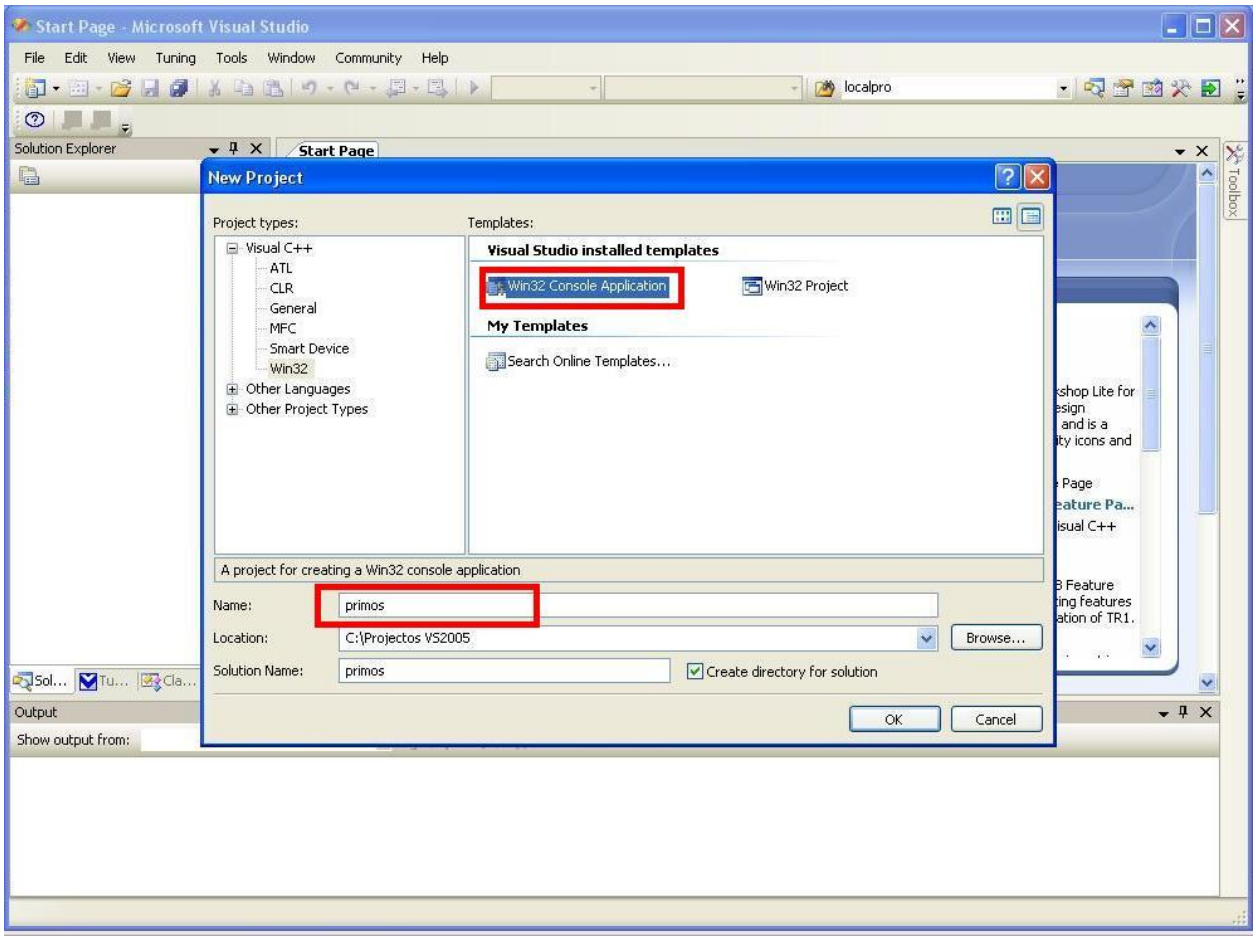
Editar y compilar un programa en Visual Studio

A continuación se muestra una breve guía para editar y compilar un programa desde Visual Studio 2008. En este caso el ejemplo se basa en el de los números primos por lo que es el nombre que se usa para los proyectos y archibos, pero este puede usarse con cualquier programa.

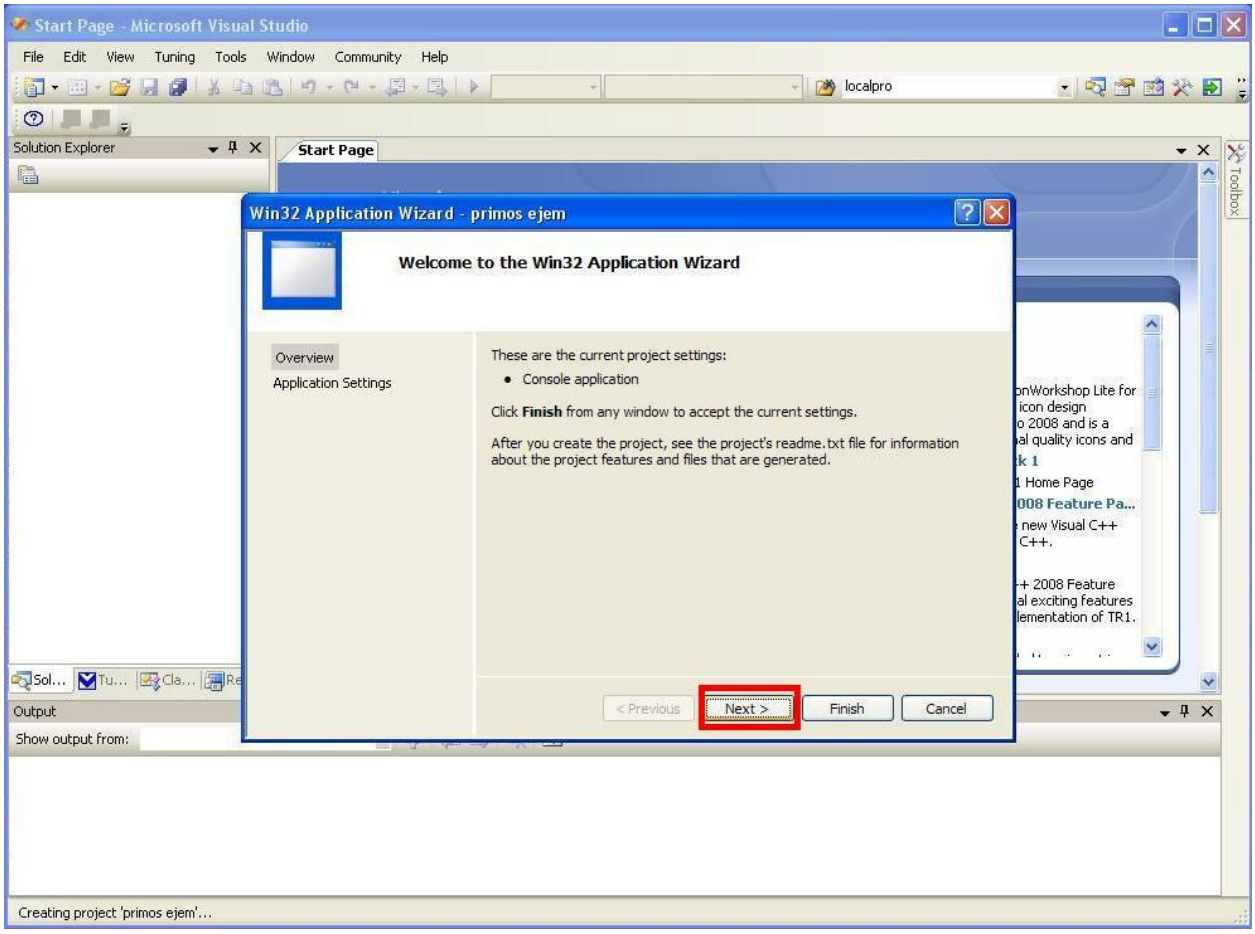
1.- Entrar a Visual Studio 2005/2008. En el menú *File* seleccionar *New* y después *Project*.



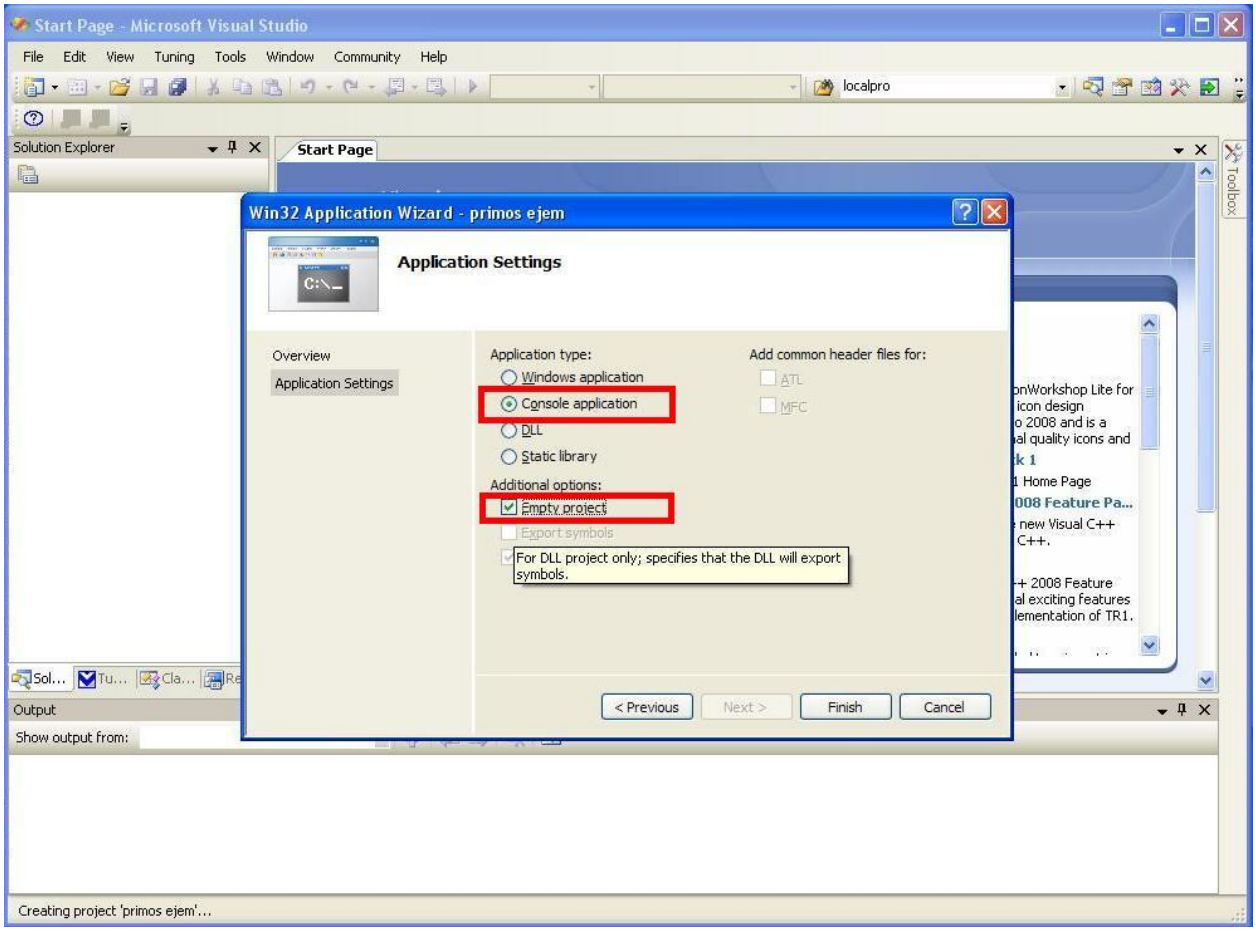
2.- Seleccionar *Win32 Console Application* y en la casilla *Name* introducir el nombre que tendrá el proyecto



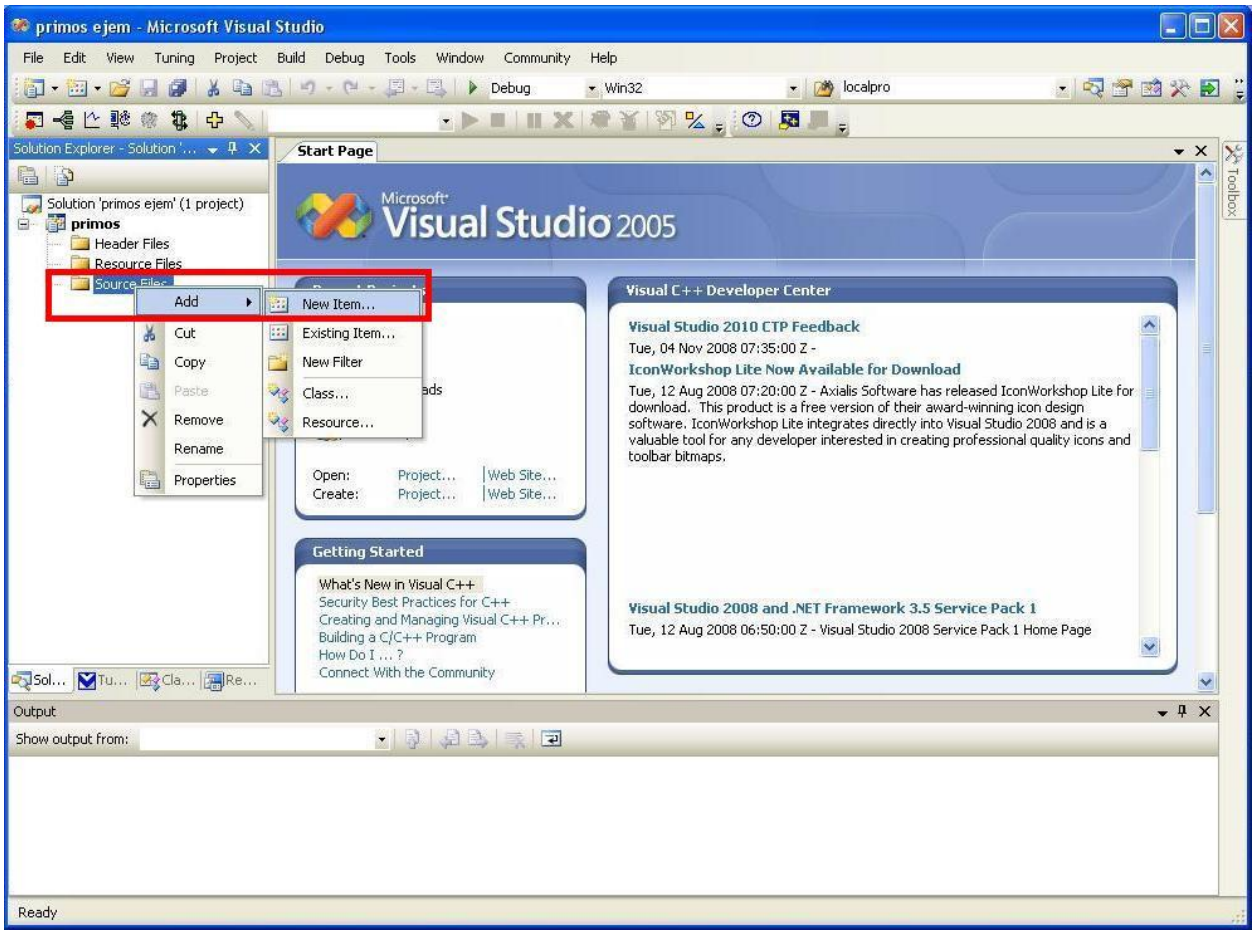
3.- Seleccionar el botón *Next*



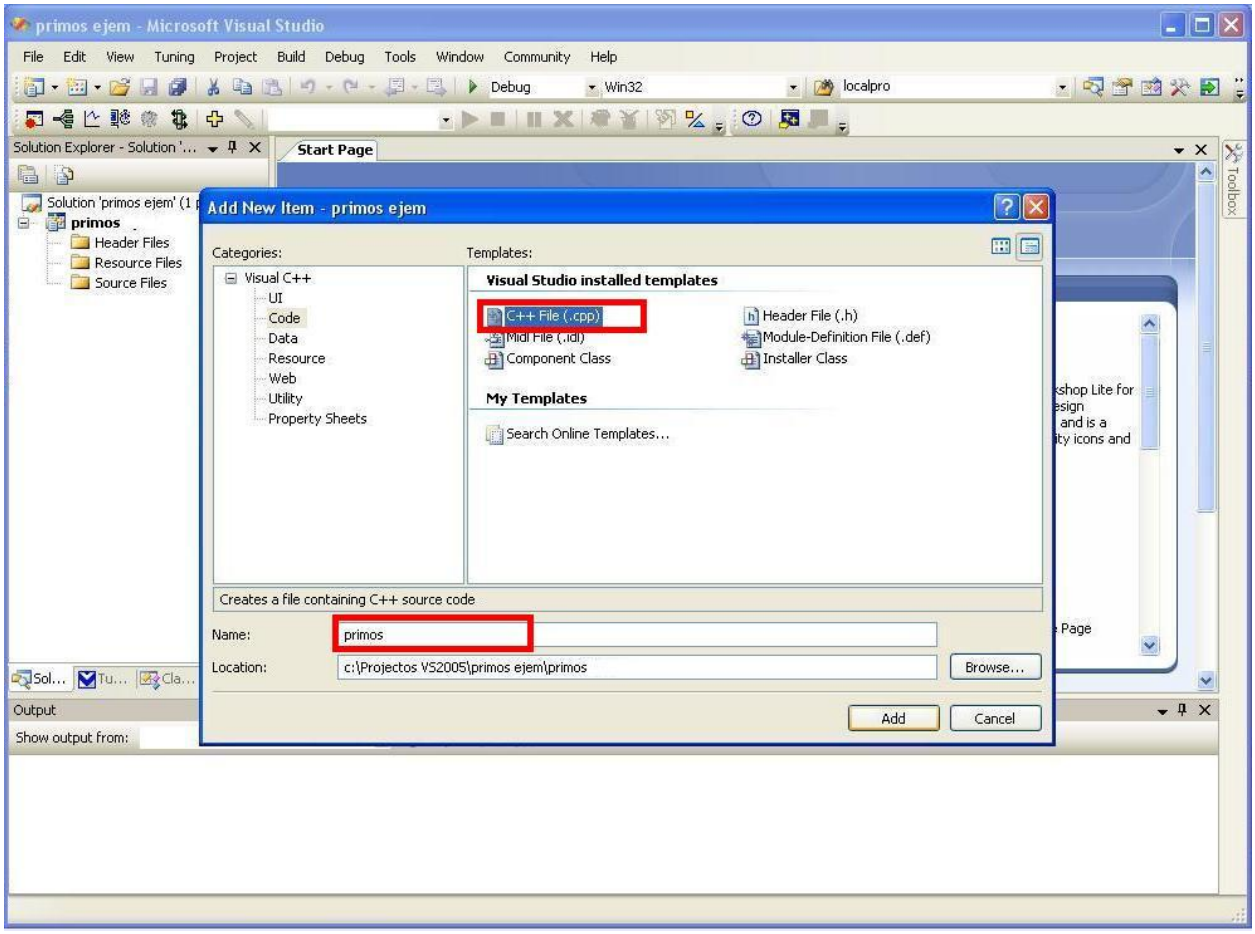
4.- Seleccionar *Console Application* y *Empty Project*, terminar presionando el botón *Finish*.



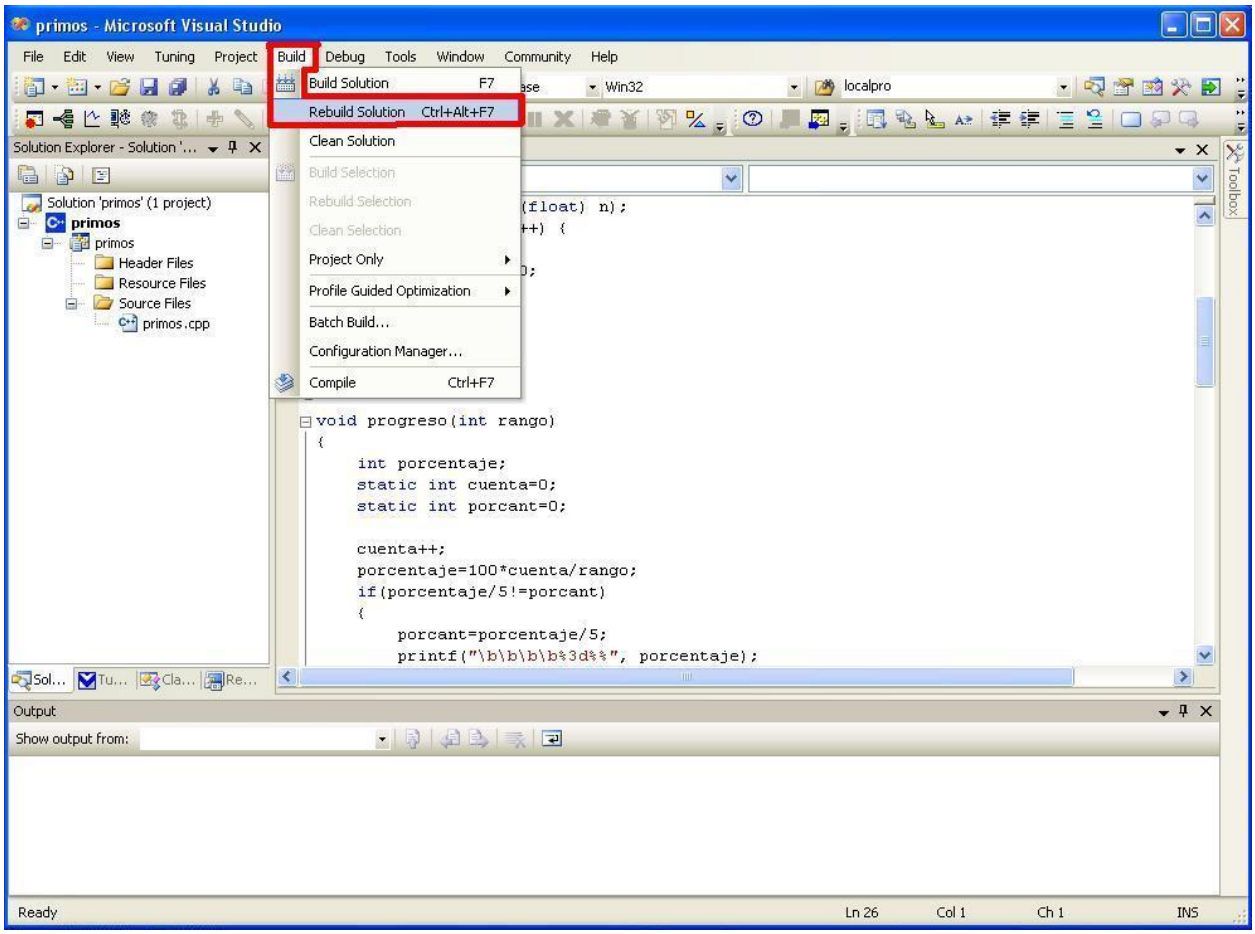
5.- Añadir el archivo fuente. Seleccionar *Source Files* y presionar el botón derecho del ratón. En el menú que se muestra a continuación seleccionar *Add* y posteriormente *New Item*.



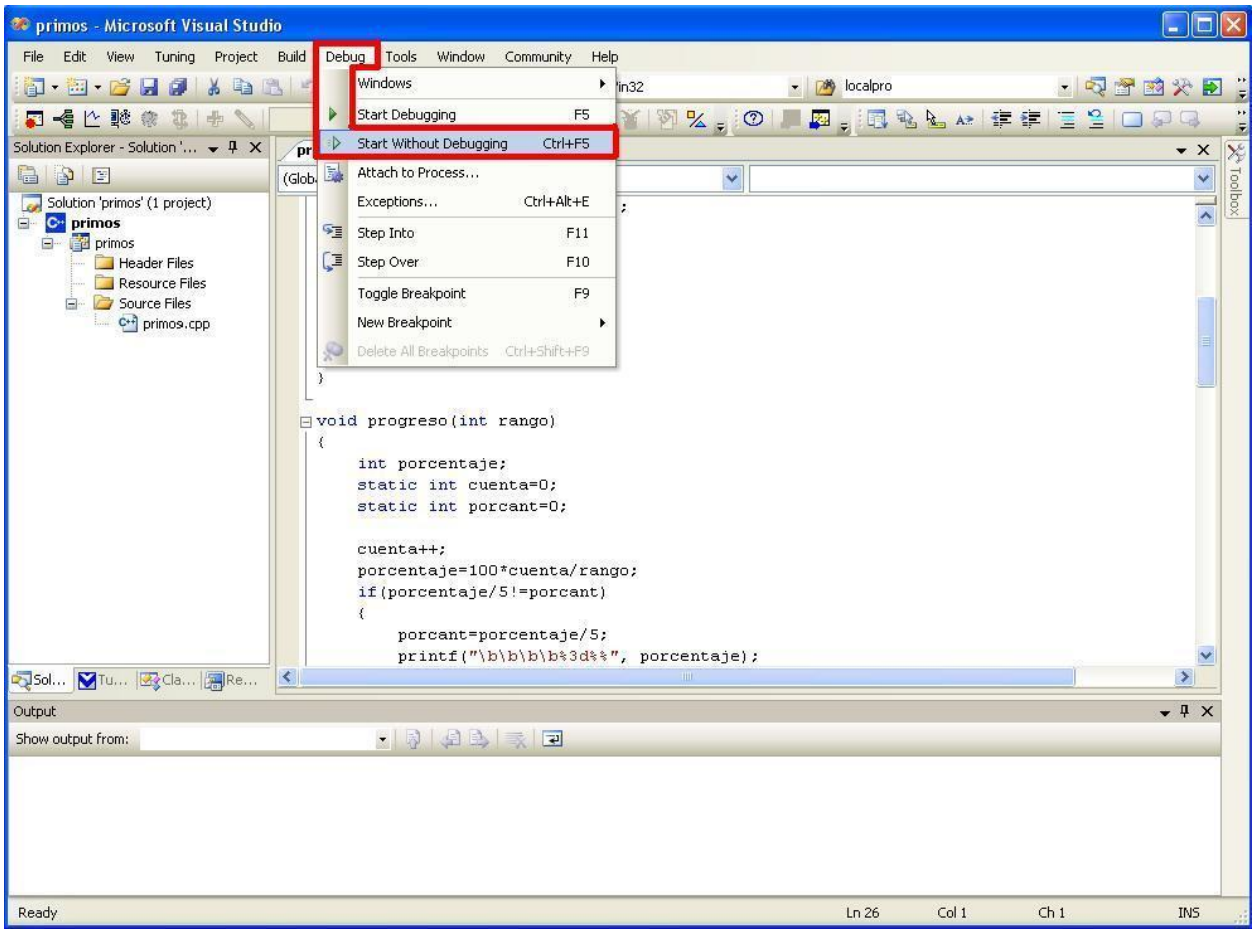
6.- Seleccionar archivo: *C++ File (.cpp)* y en la casilla *Name* nombrar el archivo



7.- Para compilar puede hacerlo presionando *<Ctrl><Alt><F7>* o seleccionar en los menús *Build -> Rebuild Solution*.



8.- Para ejecutarlo puede hacerlo presionando *<Ctrl><F5>* o desde el menú *Debug->Start Without Debugging*



Inscripción y Participación

La inscripción al Reto Intel® será enviando un correo electrónico a **retointel@academiamulticore.org** con el tema "Inscripción" y en el cuerpo del mensaje su nombre completo y universidad de la que proviene. **La hora límite para recibir inscripciones son las 11:00 AM del 19 de Noviembre de 2011**, recibirá un mensaje de confirmación de la inscripción y la lista de inscritos estará publicada en **http://academiamulticore.org/blog**.

La participación en el Reto Intel® 8 es **Individual**, por lo que no se permite ningún tipo de colaboración. El jurado se reserva el derecho de descalificar aquellas soluciones donde considere que hubo colaboración.

Entrega

La entrega de la solución será **enviando únicamente el código fuente** a través de correo electrónico **retointel@academiamulticore.org**. La hora límite para entregar la solución es las 18:00 hrs del 19 de Noviembre de 2011.

Criterios para determinar el ganador

El ganador será el concursante que logre más puntos basándonos en los siguientes criterios:

Pts = 20*ACELERACIÓN + 30*UCPU

Aceleración: Es la razón entre el tiempo serial de la aplicación entre el tiempo paralela. La aceleración máxima que se considerará para la evaluación será 2x en un procesador de dos núcleos.

UCPU: Utilización promedio de los CPU de acuerdo al informe presentado por Intel® Parallel Amplifier como el que se muestra en la Figura 2.

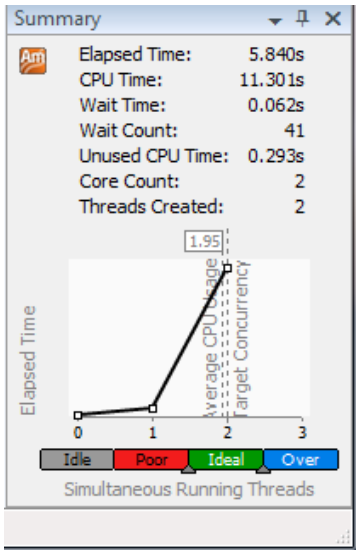


Figura 2. Utilización promedio de CPU presentada por Parallel Amplifier

El valor máximo será 2 en un procesador de dos núcleos, considerando una utilización ideal[1] del CPU.

[1] Utilización ideal se refiere a que cada CPU o núcleo ejecute un hilo. Menos hilos que núcleos es considerado como subutilización. Más hilos que núcleos es considerado como sobreutilización.

Si se presenta igualdad en los puntajes de dos o más soluciones, el criterio de desempate es tiempo de entrega.

El jurado se reserva el derecho de descalificar aquellos proyectos que presenten errores en los resultados ejecución o presenten errores potenciales en la ejecución paralela[2].

El jurado se reserva el derecho de descalificar a todos aquellos participantes que no cumplan con las normas establecidas en el concurso.

La decisión del jurado es final e inapelable.

[2] Los errores potenciales en la ejecución paralela pueden ser detectados mediante la herramienta Intel® Parallel Inspector, aunque estos no ocurran durante la ejecución.