# Lenguaje C, tercer bloque: Funciones

José Otero

[1]Departamento de informática
Universidad de Oviedo

28 de noviembre de 2007



Índice

1 Tipo puntero
- Concepto de puntero
- Operador dirección
- Operador indirección

2 Concepto de función

3 Definición, declaración y uso de funciones
- Definición de funciones
- Declaración de funciones
- Uso de funciones
- Impacto del orden de las definiciones/declaraciones
- Paso por valor
- Paso por referencia
- Llamadas a funciones desde funciones



- Las variables se almacenan en la memoria.
- La posición dentro de la memoria se identifica con un número.
- Los distintos tipos de datos:
  - Se codifican de forma distinta.
  - Ocupan distinta cantidad de memoria.
- Si se quiere acceder a los datos almacenados en la memoria *directamente*, mediante su dirección, es necesario saber cual es el tipo del objeto.
- Por eso existe el tipo de dato puntero a `int`, `char`, `float`,...
- Declaración:

```
//nombre es un puntero a tipo
tipo *nombre;
```



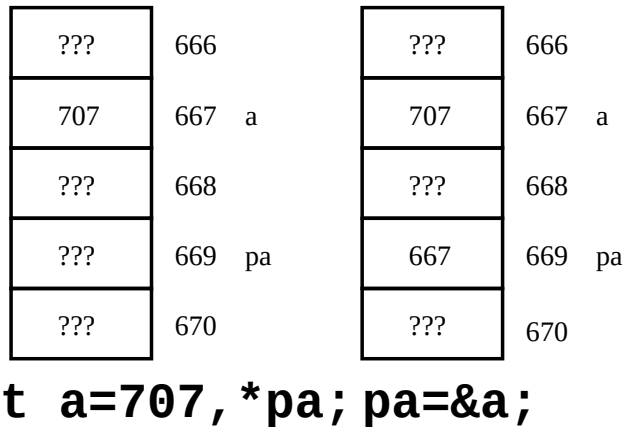
- En este curso casi siempre vamos a utilizar punteros en la forma de *variables*.
- En general, un puntero a `int`/`float`/`char`/ ... puede decirse que es una *expresión* cuyo *valor* es la dirección de memoria en donde se guarda una variable del tipo correspondiente.



- No es posible decidir *dónde* se va a almacenar un dato en la memoria de un ordenador, lo gestiona el sistema operativo.
- Si se puede averiguar en donde está almacenado un dato.
- El operador dirección es &. Aplicado a una variable, devuelve la dirección de memoria dónde se ha almacenado dicha variable.
  - Su valor es de tipo puntero al tipo de la variable.

```
//a es int, pa es puntero a int
int a=707,*pa;
//&a es puntero a int, se puede guardar en pa
pa=&a;
```

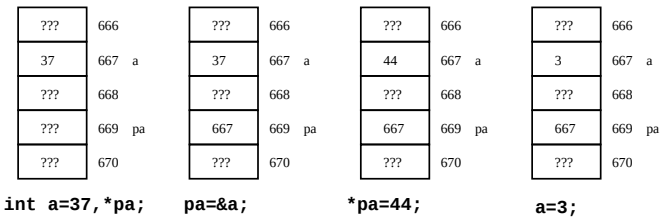




- Es posible utilizar y/o modificar el valor almacenado en una posición de memoria mediante un puntero.
- El operador indirección es `*`. Aplicado a una expresión de tipo puntero es un sinónimo de la variable contenida en esa posición.

```
int a=37,*pa;
//pa es la direccion de memoria
//en donde se almacena a
pa=&a;
*pa=44;//*pa es lo mismo que a
printf("\n%d",a);//muestra 44
a=3;
printf("\n%d",*pa);//muestra 3
```








- Una función es una colección independiente de declaraciones y sentencias, generalmente enfocadas a realizar una tarea específica.
- Un programa en C consta al menos de una función, `main()`.
- El problema general se puede descomponer en funciones, más fáciles de codificar y de mantener.
- La ejecución de un programa comienza por la función `main`.



- Como cualquier objeto en C, una función debe declararse y definirse antes de ser utilizada.
- La declaración y/o definición no tiene por que hacerse en el mismo fichero fuente de `main`.
- En este curso ya se han utilizado funciones: `printf`, `scanf`, `sqrt`, `system`...
  - Esas funciones estaban declaradas en los ficheros de cabeceras que se incluían al principio de los programas.



La definición de una función consta de:

- Tipo de retorno. Alguno de los vistos: `void`, `int`, `float`, `char`, ...
- Nombre. Es un identificador, al igual que el nombre de una variable.
- Lista de parámetros formales, entre `()`. Variables en las que se almacenan los datos con los que operará la función. Se especifica su tipo y nombre. Una lista vacía se representa por `void`.
- Cuerpo. Encerrado entre `{}`, código fuente que reúne las instrucciones que realizan el cálculo correspondiente.
  - Además, si la función *no* es `void`, incluye la sentencia `return` que representa la acción de devolver el resultado calculado.



Esquema:

```
tipof nombref(tipo1 nombre1, tipo2 nombre2, ...)
{
//declaraciones locales
...
//sentencias
...
return expresion_tipof;
}
```

Importante:

- Las variables que se declaran en el cuerpo de la función son locales.
- Los parámetros formales también son locales a la función.
- Se *crean cada vez* que se ejecuta la función y se *destruyen* cuando finaliza la función.
- Pueden escribirse varias sentencias `return`.
  - Sólo se puede devolver *un* valor, el de la expresión que se escribe después de `return`.
  - La función termina cuando se ejecuta la primera de ellas.
- Si no hay `return` la función termina cuando se alcanza la `}` final.



La declaración de una función (prototipo) sirve para que el compilador compruebe si se usa correctamente.
Consta de:

- Tipo de retorno. Alguno de los vistos: `void`, `int`, `float`, `char`, ...
- Nombre. Es un identificador, al igual que el nombre de una variable.
- Tipo de los parámetros formales, encerrados entre `()`, separados por comas. Aunque se especifique el nombre el compilador lo ignora.
- Se termina con un `;`.

NOTA:

- Si la definición de una función se escribe *antes* (en el mismo fuente) de ser usada, no hace falta declararla.



- Cuando una función se usa, se está realizando una *llamada* a esa función. La llamada puede hacerse:
  - Desde la función principal `main`.
  - Desde otras funciones.
  - Incluso desde la misma función.
- Una función puede ser llamada varias veces.
- La llamada consiste en:
  - Nombre de la función.
  - Lista de parámetros reales, separados por comas, entre `()`.
    - *Expresiones* iguales en tipo, número y orden a los parámetros formales. Si está vacía no se escribe nada entre los `()`.

```
...nombref(param1, param2, ...)...
```



Importante:

- Si la función *no* es `void`, la llamada se puede escribir en cualquier parte del programa en donde se pueda escribir una expresión del tipo que devuelve.
- Si la función *es* (devuelve) `void` la llamada forma una única sentencia acabada en `;`.
- Si la función no recibe ningún parámetro, se escriben los `()` sin contenido dentro de los mismos.



El tipo de retorno no es `void` y la lista de parámetros no está vacía.

```
tipof nombref(tipo1 arg1, tipo2 arg2, ...)
{
...
//al menos un return
return exp_tipo;
}
int main()
{
...
...nombref(exp_tipo1, exp_tipo2, ...)...
...
}
```



El tipo de retorno es `void` y la lista de parámetros no está vacía.

```
void nombref(tipo1 arg1, tipo2 arg2, ...)
{
...
//sin return
}
int main()
{
...
nombref(exp_tipo1, exp_tipo2, ...);
...
}
```

El tipo de retorno no es `void` y la lista de parámetros está vacía.

```
tipof nombref(void)
{
...
//al menos un return
return exp_tipo;
}
int main()
{
...
...nombref()...
...
}
```



El tipo de retorno es `void` y la lista de parámetros está vacía.

```
void nombref(void)
{
...
}
int main()
{
...
nombref();
...
}
```



No siempre es adecuado definir y declarar las funciones *simultáneamente*, las funciones *tienen* que definirse en orden.

```
#include<stdio.h>
//esta se define primero
void muestra_mensaje(void)
{
  printf("\nEntero positivo?");
}
//porque se usa aqui
int lee_positivo(void)
{
  int n;
  do{
    muestra_mensaje();
    scanf("%d",&n);
  }while(n<0);
  return n;
}

int main()
{
  int a;
  a=lee_positivo();
  printf("\n%d",a);
}
```

```
#include<stdio.h>
//declaramos las funciones en
//en cualquier orden
void muestra_mensaje(void);
int lee_positivo(void);
int main()
{
  int a;
  a=lee_positivo();
  printf("\n%d",a);
}
//las definimos en cualquier orden
int lee_positivo(void)
{
  int n;
  do{
    muestra_mensaje();
    scanf("%d",&n);
  }while(n<0);
  return n;
}

void muestra_mensaje(void)
{
  printf("\nEntero positivo?");
}
```

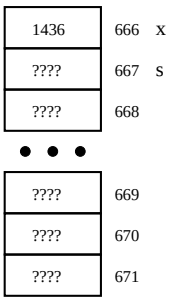


Cuando los parámetros formales son de alguno de los tipos: `int`, `float`, `char`,... se están pasando los parámetros por *valor*.

- Los parámetros reales *pueden* ser expresiones (del mismo tipo que el correspondiente parámetro formal).
- Su *valor* se *copia* en el correspondiente parámetro formal.
- Aunque se modifiquen los parámetros formales, dentro de la función, no se modifican los parámetros reales.

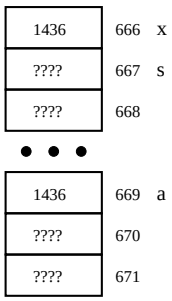


```
#include<stdio.h>
int suma_cifras(int a)
{
  int suma=0;
  while(a!=0)
    {
      suma+=a%10;
      a/=10;
    }
  return suma;
}
int main()
{
  int x=1436,s;
  s=suma_cifras(x);
  printf("\n%d",s);
}
```

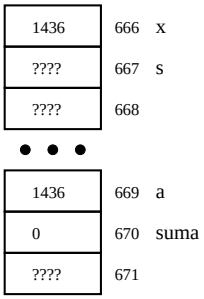





```
#include<stdio.h>
int suma_cifras(int a)
{
  int suma=0;
  while(a!=0)
    {
      suma+=a%10;
      a/=10;
    }
  return suma;
}
int main()
{
  int x=1436,s;
  s=suma_cifras(x);
  printf("\n%d",s);
}
```

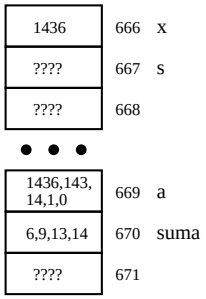


```c
#include<stdio.h>
int suma_cifras(int a)
{
  int suma=0;
  while(a!=0)
    {
      suma+=a%10;
      a/=10;
    }
  return suma;
}
int main()
{
  int x=1436,s;
  s=suma_cifras(x);
  printf("\n%d",s);
}
```



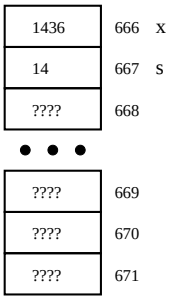



```c
#include<stdio.h>
int suma_cifras(int a)
{
  int suma=0;
  while(a!=0)
    {
      suma+=a%10;
      a/=10;
    }
  return suma;
}
int main()
{
  int x=1436,s;
  s=suma_cifras(x);
  printf("\n%d",s);
}
```






```c
#include<stdio.h>
int suma_cifras(int a)
{
  int suma=0;
  while(a!=0)
    {
      suma+=a%10;
      a/=10;
    }
  return suma;
}
int main()
{
  int x=1436,s;
  s=suma_cifras(x);
  printf("\n%d",s);
}
```

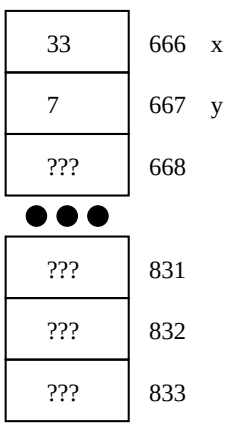





- Cuando alguno de los parámetros formales de una función es de tipo puntero, se está utilizando el paso por referencia.
- La función puede modificar el valor almacenado en las posiciones de memoria que se pasan como punteros.
- Se usa el paso por referencia cuando:
  - Se desea que algún parámetro sea de entrada y salida.
  - No se desea hacer copias de algún parámetro, por cuestiones de eficiencia (tiempo y/o espacio).
  - De forma implícita, cuando los datos que maneja la función son punteros.

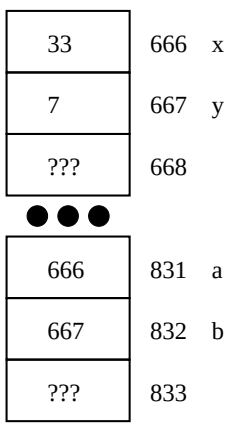


```c
#include<stdio.h>
void intercambia(int *a,int *b)
{
int tmp;
tmp=*a;
*a=*b;
*b=tmp;
}
int main()
{
int x=33,y=7;
intercambia(&x,&y);
printf("\n%d %d",x,y);
}
```

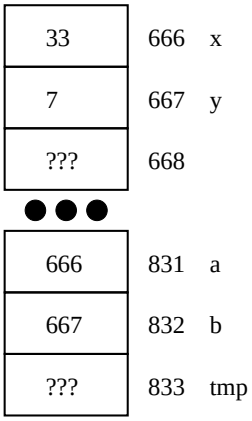





```c
#include<stdio.h>
void intercambia(int *a,int *b)
{
int tmp;
tmp=*a;
*a=*b;
*b=tmp;
}
int main()
{
int x=33,y=7;
intercambia(&x,&y);
printf("\n%d %d",x,y);
}
```

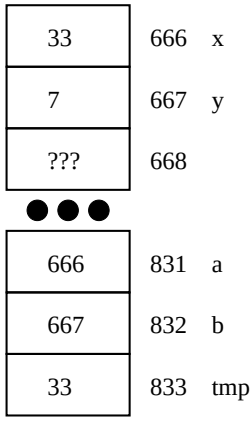


```
#include<stdio.h>
void intercambia(int *a,int *b)
{
int tmp;
tmp=*a;
*a=*b;
*b=tmp;
}
int main()
{
int x=33,y=7;
intercambia(&x,&y);
printf("\n%d %d",x,y);
}
```



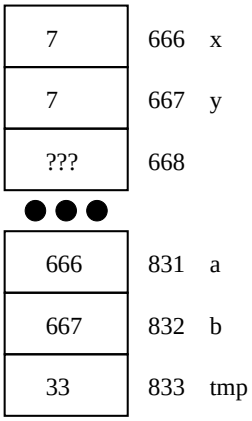



```
#include<stdio.h>
void intercambia(int *a,int *b)
{
int tmp;
tmp=*a;
*a=*b;
*b=tmp;
}
int main()
{
int x=33,y=7;
intercambia(&x,&y);
printf("\n%d %d",x,y);
}
```



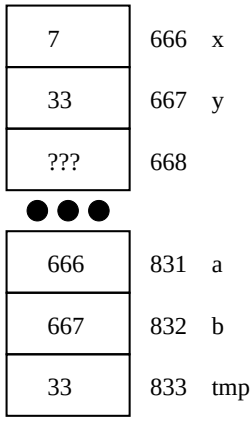



```
#include<stdio.h>
void intercambia(int *a,int *b)
{
int tmp;
tmp=*a;
*a=*b;
*b=tmp;
}
int main()
{
int x=33,y=7;
intercambia(&x,&y);
printf("\n%d %d",x,y);
}
```



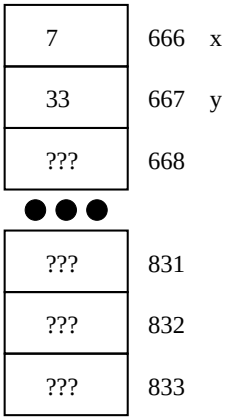



```
#include<stdio.h>
void intercambia(int *a,int *b)
{
int tmp;
tmp=*a;
*a=*b;
*b=tmp;
}
int main()
{
int x=33,y=7;
intercambia(&x,&y);
printf("\n%d %d",x,y);
}
```






```
#include<stdio.h>
void intercambia(int *a,int *b)
{
int tmp;
tmp=*a;
*a=*b;
*b=tmp;
}
int main()
{
int x=33,y=7;
intercambia(&x,&y);
printf("\n%d %d",x,y);
}
```






- Se pueden hacer llamadas a funciones desde otras funciones (además de desde `main`).
- Se pueden hacer varias llamadas a la misma función.
  - Desde el mismo bloque.
  - Desde otros bloques.
- Los parámetros formales van tomando el valor de los parámetros reales de cada llamada.
- Los parámetros formales y las variables locales se crean al principio de la llamada y se destruyen al finalizar esta.

```c
#include<stdio.h>
int fact(int num)
{
  int f=1,i;
  for (i=2;i<=num;i++)
    f=f*i;
  return f;
}
int ncomb(int n, int m)
{
  int fn,fm,fnm;
  fn=fact(n);
  fm=fact(m);
  fnm=fact(n-m);
  return fn/fm/fnm;
}
int main()
{
  int a,b,c;
  scanf("%d%d",&a,&b);
  c=ncomb(a,b);
  printf("\n%d",c);
}
```

| | | |
|---|---|---|
| ??? | 666 | a |
| ??? | 667 | b |
| ??? | 668 | c |
| ??? | 669 | |
| ??? | 670 | |
| ??? | 671 | |
| ??? | 672 | |
| ??? | 673 | |
| ??? | 674 | |
| ??? | 675 | |
| ??? | 676 | |
| ??? | 677 | |



```c
#include<stdio.h>
int fact(int num)
{
  int f=1,i;
  for (i=2;i<=num;i++)
    f=f*i;
  return f;
}
int ncomb(int n, int m)
{
  int fn,fm,fnm;
  fn=fact(n);
  fm=fact(m);
  fnm=fact(n-m);
  return fn/fm/fnm;
}
int main()
{
  int a,b,c;
  scanf("%d%d",&a,&b);
  c=ncomb(a,b);
  printf("\n%d",c);
}
```

| | | | |
|---|---|---|---|
| 5 | 666 | a | El usuario |
| 3 | 667 | b | teclea 5 y 3 |
| ??? | 668 | c | |
| ??? | 669 | | |
| ??? | 670 | | |
| ??? | 671 | | |
| ??? | 672 | | |
| ??? | 673 | | |
| ??? | 674 | | |
| ??? | 675 | | |
| ??? | 676 | | |
| ??? | 677 | | |



```c
#include<stdio.h>
int fact(int num)
{
  int f=1,i;
  for (i=2;i<=num;i++)
    f=f*i;
  return f;
}
int ncomb(int n, int m)
{
  int fn,fm,fnm;
  fn=fact(n);
  fm=fact(m);
  fnm=fact(n-m);
  return fn/fm/fnm;
}
int main()
{
  int a,b,c;
  scanf("%d%d",&a,&b);
  c=ncomb(a,b);
  printf("\n%d",c);
}
```

| | | |
|---|---|---|
| 5 | 666 | a |
| 3 | 667 | b |
| ??? | 668 | c |
| ??? | 669 | |
| 5 | 670 | n |
| 3 | 671 | m |
| ??? | 672 | |
| ??? | 673 | |
| ??? | 674 | |
| ??? | 675 | |
| ??? | 676 | |
| ??? | 677 | |



```c
#include<stdio.h>
int fact(int num)
{
  int f=1,i;
  for (i=2;i<=num;i++)
    f=f*i;
  return f;
}
int ncomb(int n, int m)
{
  int fn,fm,fnm;
  fn=fact(n);
  fm=fact(m);
  fnm=fact(n-m);
  return fn/fm/fnm;
}
int main()
{
  int a,b,c;
  scanf("%d%d",&a,&b);
  c=ncomb(a,b);
  printf("\n%d",c);
}
```

| | | |
|---|---|---|
| 5 | 666 | a |
| 3 | 667 | b |
| ??? | 668 | c |
| ??? | 669 | |
| 5 | 670 | n |
| 3 | 671 | m |
| ??? | 672 | fn |
| ??? | 673 | fm |
| ??? | 674 | fnm |
| ??? | 675 | |
| ??? | 676 | |
| ??? | 677 | |



```c
#include<stdio.h>
int fact(int num)
{
  int f=1,i;
  for (i=2;i<=num;i++)
    f=f*i;
  return f;
}
int ncomb(int n, int m)
{
  int fn,fm,fnm;
  fn=fact(n);
  fm=fact(m);
  fnm=fact(n-m);
  return fn/fm/fnm;
}
int main()
{
  int a,b,c;
  scanf("%d%d",&a,&b);
  c=ncomb(a,b);
  printf("\n%d",c);
}
```

| | | |
|---|---|---|
| 5 | 666 | a |
| 3 | 667 | b |
| ??? | 668 | c |
| ??? | 669 | |
| 5 | 670 | n |
| 3 | 671 | m |
| ??? | 672 | fn |
| ??? | 673 | fm |
| ??? | 674 | fnm |
| 5 | 675 | num |
| ??? | 676 | |
| ??? | 677 | |



```c
#include<stdio.h>
int fact(int num)
{
  int f=1,i;
  for (i=2;i<=num;i++)
    f=f*i;
  return f;
}
int ncomb(int n, int m)
{
  int fn,fm,fnm;
  fn=fact(n);
  fm=fact(m);
  fnm=fact(n-m);
  return fn/fm/fnm;
}
int main()
{
  int a,b,c;
  scanf("%d%d",&a,&b);
  c=ncomb(a,b);
  printf("\n%d",c);
}
```

| | | |
|---|---|---|
| 5 | 666 | a |
| 3 | 667 | b |
| ??? | 668 | c |
| ??? | 669 | |
| 5 | 670 | n |
| 3 | 671 | m |
| ??? | 672 | fn |
| ??? | 673 | fm |
| ??? | 674 | fnm |
| 5 | 675 | num |
| 1 | 676 | f |
| ??? | 677 | i |

```c
#include<stdio.h>
int fact(int num)
{
  int f=1,i;
  for (i=2;i<=num;i++)
    f=f*i;
  return f;
}
int ncomb(int n, int m)
{
  int fn,fm,fnm;
  fn=fact(n);
  fm=fact(m);
  fnm=fact(n-m);
  return fn/fm/fnm;
}
int main()
{
  int a,b,c;
  scanf("%d%d",&a,&b);
  c=ncomb(a,b);
  printf("\n%d",c);
}
```

| | | |
|---|---|---|
| 5 | 666 | a |
| 3 | 667 | b |
| ??? | 668 | c |
| ??? | 669 | |
| 5 | 670 | n |
| 3 | 671 | m |
| ??? | 672 | fn |
| ??? | 673 | fm |
| ??? | 674 | fnm |
| 5 | 675 | num |
| 2,6,24,120 | 676 | f |
| 2,3,4,5 | 677 | i |



```c
#include<stdio.h>
int fact(int num)
{
  int f=1,i;
  for (i=2;i<=num;i++)
    f=f*i;
  return f;
}
int ncomb(int n, int m)
{
  int fn,fm,fnm;
  fn=fact(n);
  fm=fact(m);
  fnm=fact(n-m);
  return fn/fm/fnm;
}
int main()
{
  int a,b,c;
  scanf("%d%d",&a,&b);
  c=ncomb(a,b);
  printf("\n%d",c);
}
```

| | | |
|---|---|---|
| 5 | 666 | a |
| 3 | 667 | b |
| ??? | 668 | c |
| ??? | 669 | |
| 5 | 670 | n |
| 3 | 671 | m |
| 120 | 672 | fn |
| ??? | 673 | fm |
| ??? | 674 | fnm |
| 5 | 675 | num |
| ??? | 676 | |
| ??? | 677 | |



```c
#include<stdio.h>
int fact(int num)
{
  int f=1,i;
  for (i=2;i<=num;i++)
    f=f*i;
  return f;
}
int ncomb(int n, int m)
{
  int fn,fm,fnm;
  fn=fact(n);
  fm=fact(m);
  fnm=fact(n-m);
  return fn/fm/fnm;
}
int main()
{
  int a,b,c;
  scanf("%d%d",&a,&b);
  c=ncomb(a,b);
  printf("\n%d",c);
}
```

| | | |
|---|---|---|
| 5 | 666 | a |
| 3 | 667 | b |
| ??? | 668 | c |
| ??? | 669 | |
| 5 | 670 | n |
| 3 | 671 | m |
| 120 | 672 | fn |
| ??? | 673 | fm |
| ??? | 674 | fnm |
| 3 | 675 | num |
| 1 | 676 | f |
| ??? | 677 | i |



```c
#include<stdio.h>
int fact(int num)
{
  int f=1,i;
  for (i=2;i<=num;i++)
    f=f*i;
  return f;
}
int ncomb(int n, int m)
{
  int fn,fm,fnm;
  fn=fact(n);
  fm=fact(m);
  fnm=fact(n-m);
  return fn/fm/fnm;
}
int main()
{
  int a,b,c;
  scanf("%d%d",&a,&b);
  c=ncomb(a,b);
  printf("\n%d",c);
}
```

| | | |
|---|---|---|
| 5 | 666 | a |
| 3 | 667 | b |
| ??? | 668 | c |
| ??? | 669 | |
| 5 | 670 | n |
| 3 | 671 | m |
| 120 | 672 | fn |
| ??? | 673 | fm |
| ??? | 674 | fnm |
| 3 | 675 | num |
| 2,6 | 676 | f |
| 2,3 | 677 | i |



```c
#include<stdio.h>
int fact(int num)
{
  int f=1,i;
  for (i=2;i<=num;i++)
    f=f*i;
  return f;
}
int ncomb(int n, int m)
{
  int fn,fm,fnm;
  fn=fact(n);
  fm=fact(m);
  fnm=fact(n-m);
  return fn/fm/fnm;
}
int main()
{
  int a,b,c;
  scanf("%d%d",&a,&b);
  c=ncomb(a,b);
  printf("\n%d",c);
}
```

| | | |
|---|---|---|
| 5 | 666 | a |
| 3 | 667 | b |
| ??? | 668 | c |
| ??? | 669 | |
| 5 | 670 | n |
| 3 | 671 | m |
| 120 | 672 | fn |
| 6 | 673 | fm |
| ??? | 674 | fnm |
| ??? | 675 | |
| ??? | 676 | |
| ??? | 677 | |

```c
#include<stdio.h>
int fact(int num)
{
  int f=1,i;
  for (i=2;i<=num;i++)
    f=f*i;
  return f;
}
int ncomb(int n, int m)
{
  int fn,fm,fnm;
  fn=fact(n);
  fm=fact(m);
  fnm=fact(n-m);
  return fn/fm/fnm;
}
int main()
{
  int a,b,c;
  scanf("%d%d",&a,&b);
  c=ncomb(a,b);
  printf("\n%d",c);
}
```

| | | |
|---|---|---|
| 5 | 666 | a |
| 3 | 667 | b |
| ??? | 668 | c |
| ??? | 669 | |
| 5 | 670 | n |
| 3 | 671 | m |
| 120 | 672 | fn |
| 6 | 673 | fm |
| ??? | 674 | fnm |
| 2 | 675 | num |
| ??? | 676 | |
| ??? | 677 | |



```c
#include<stdio.h>
int fact(int num)
{
  int f=1,i;
  for (i=2;i<=num;i++)
    f=f*i;
  return f;
}
int ncomb(int n, int m)
{
  int fn,fm,fnm;
  fn=fact(n);
  fm=fact(m);
  fnm=fact(n-m);
  return fn/fm/fnm;
}
int main()
{
  int a,b,c;
  scanf("%d%d",&a,&b);
  c=ncomb(a,b);
  printf("\n%d",c);
}
```

| | | |
|---|---|---|
| 5 | 666 | a |
| 3 | 667 | b |
| ??? | 668 | c |
| ??? | 669 | |
| 5 | 670 | n |
| 3 | 671 | m |
| 120 | 672 | fn |
| 6 | 673 | fm |
| ??? | 674 | fnm |
| 2 | 675 | num |
| 1 | 676 | f |
| ??? | 677 | i |



```c
#include<stdio.h>
int fact(int num)
{
  int f=1,i;
  for (i=2;i<=num;i++)
    f=f*i;
  return f;
}
int ncomb(int n, int m)
{
  int fn,fm,fnm;
  fn=fact(n);
  fm=fact(m);
  fnm=fact(n-m);
  return fn/fm/fnm;
}
int main()
{
  int a,b,c;
  scanf("%d%d",&a,&b);
  c=ncomb(a,b);
  printf("\n%d",c);
}
```

| | | |
|---|---|---|
| 5 | 666 | a |
| 3 | 667 | b |
| ??? | 668 | c |
| ??? | 669 | |
| 5 | 670 | n |
| 3 | 671 | m |
| 120 | 672 | fn |
| 6 | 673 | fm |
| ??? | 674 | fnm |
| 2 | 675 | num |
| 2 | 676 | f |
| 2 | 677 | i |



```c
#include<stdio.h>
int fact(int num)
{
  int f=1,i;
  for (i=2;i<=num;i++)
    f=f*i;
  return f;
}
int ncomb(int n, int m)
{
  int fn,fm,fnm;
  fn=fact(n);
  fm=fact(m);
  fnm=fact(n-m);
  return fn/fm/fnm;
}
int main()
{
  int a,b,c;
  scanf("%d%d",&a,&b);
  c=ncomb(a,b);
  printf("\n%d",c);
}
```

| | | |
|---|---|---|
| 5 | 666 | a |
| 3 | 667 | b |
| ??? | 668 | c |
| ??? | 669 | |
| 5 | 670 | n |
| 3 | 671 | m |
| 120 | 672 | fn |
| 6 | 673 | fm |
| 2 | 674 | fnm |
| ??? | 675 | |
| ??? | 676 | |
| ??? | 677 | |



```c
#include<stdio.h>
int fact(int num)
{
  int f=1,i;
  for (i=2;i<=num;i++)
    f=f*i;
  return f;
}
int ncomb(int n, int m)
{
  int fn,fm,fnm;
  fn=fact(n);
  fm=fact(m);
  fnm=fact(n-m);
  return fn/fm/fnm;
}
int main()
{
  int a,b,c;
  scanf("%d%d",&a,&b);
  c=ncomb(a,b);
  printf("\n%d",c);
}
```

| | | |
|---|---|---|
| 5 | 666 | a |
| 3 | 667 | b |
| 10 | 668 | c |
| ??? | 669 | |
| ??? | 670 | |
| ??? | 671 | |
| ??? | 672 | |
| ??? | 673 | |
| ??? | 674 | |
| ??? | 675 | |
| ??? | 676 | |
| ??? | 677 | |