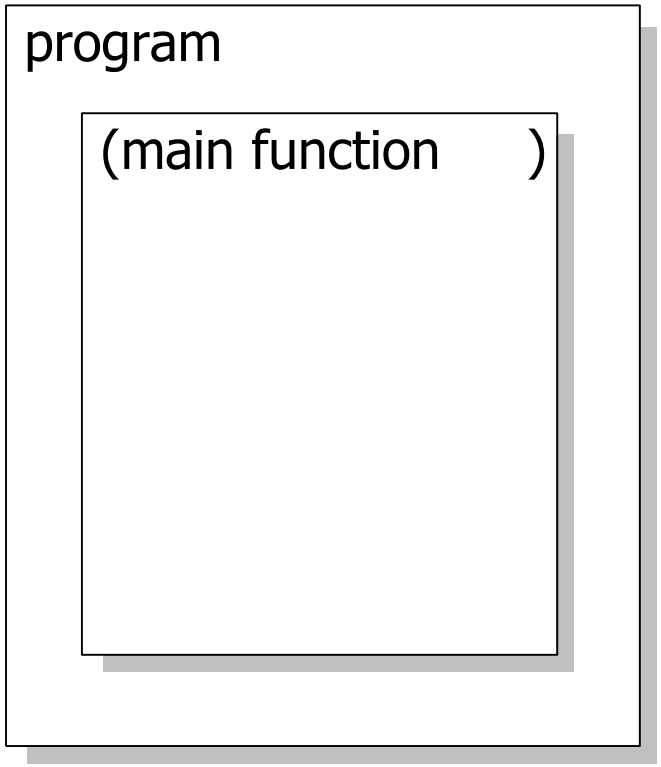
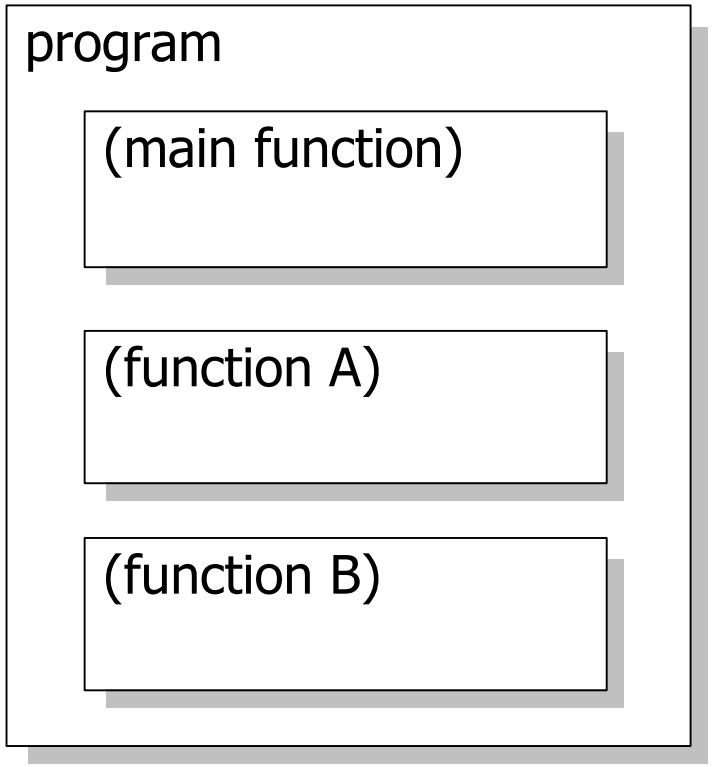
Outline

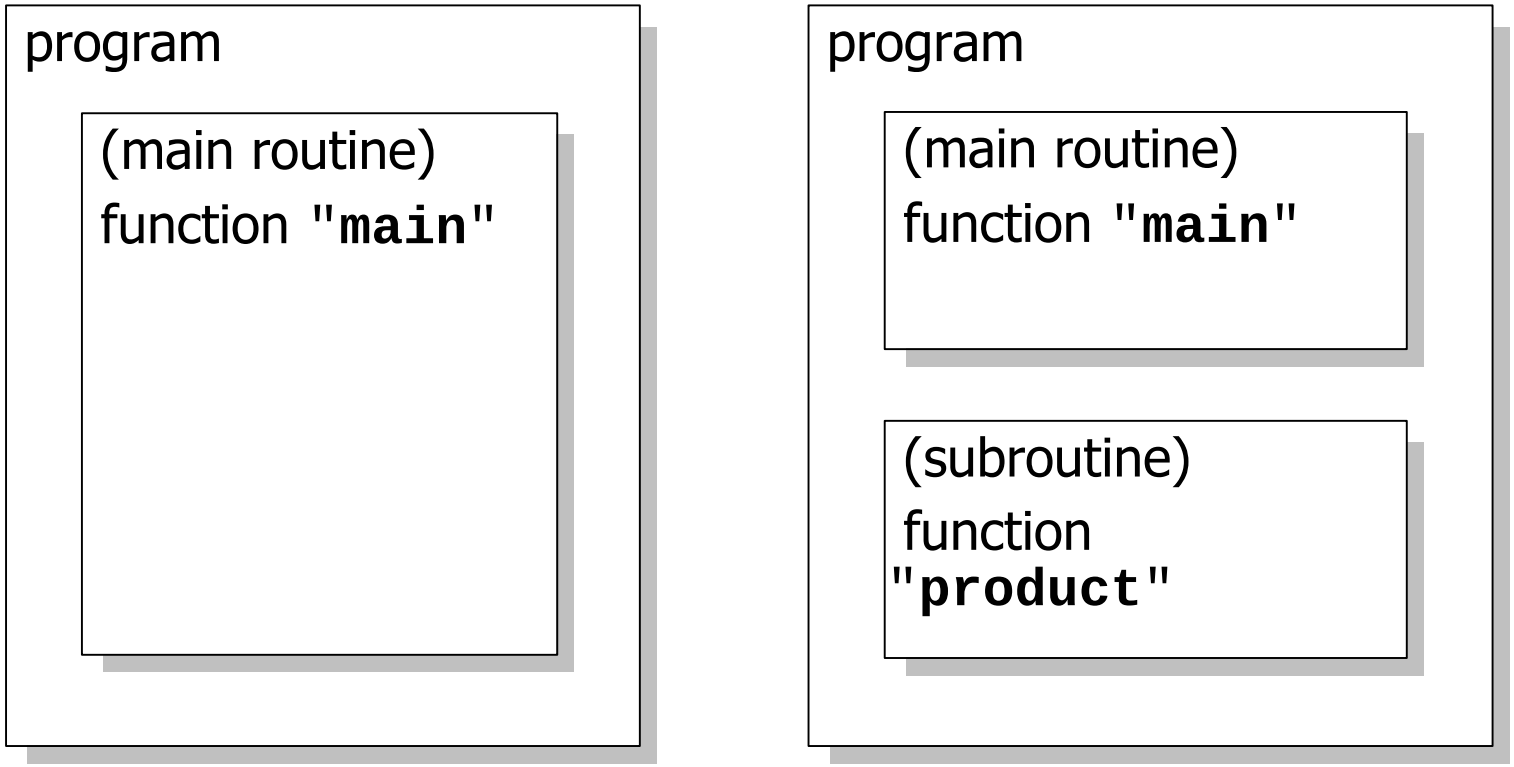
- Function and program structure
- Returning a value
- Function arguments
- Argument passing
- Local, external, and static variables
- Scope rules
- Recursion

The Structure of a C Program





Functions



A *function* is an independent section of program code that performs a certain task and has been assigned a name.

A Short C Program

```
/* Calculate the product of 2 numbers */
#include  <stdio.h>

int  a, b, c;
int product( int x, int y );

main()
{
  printf("Enter a number: ");
  scanf( "%d", &a );
  printf("Enter another number: ");
  scanf( "%d", &b );
  c = product( a, b );
  printf( "%d times %d = %d\n", a, b, c );
}
int product( int x, int y )
{
  return( x * y );
}
```

Function : What is it

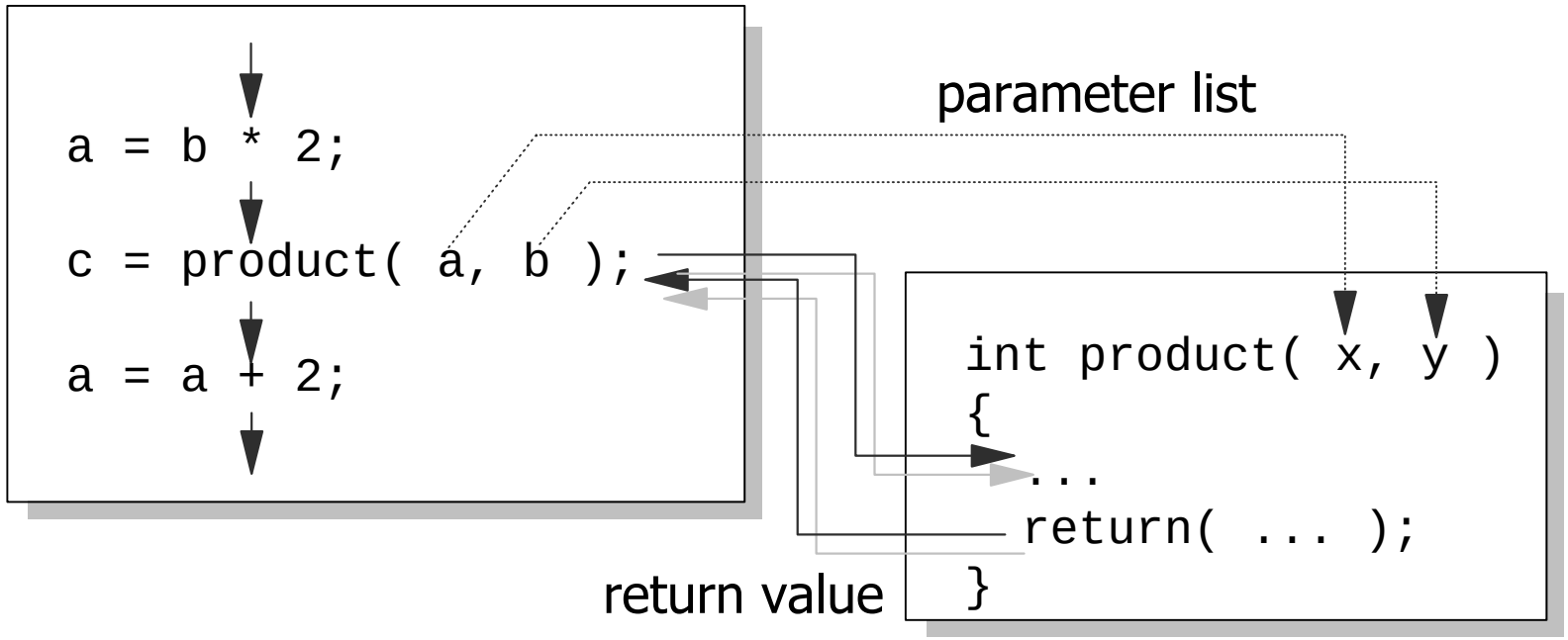
- Each function has a unique name.
- A function is not executed until it is called.
- A function performs a specific task.
- A function can return a value to the calling program.

```
PrintHeading();
InverseMatrix( A, InvA );
len = strlen( s );
```

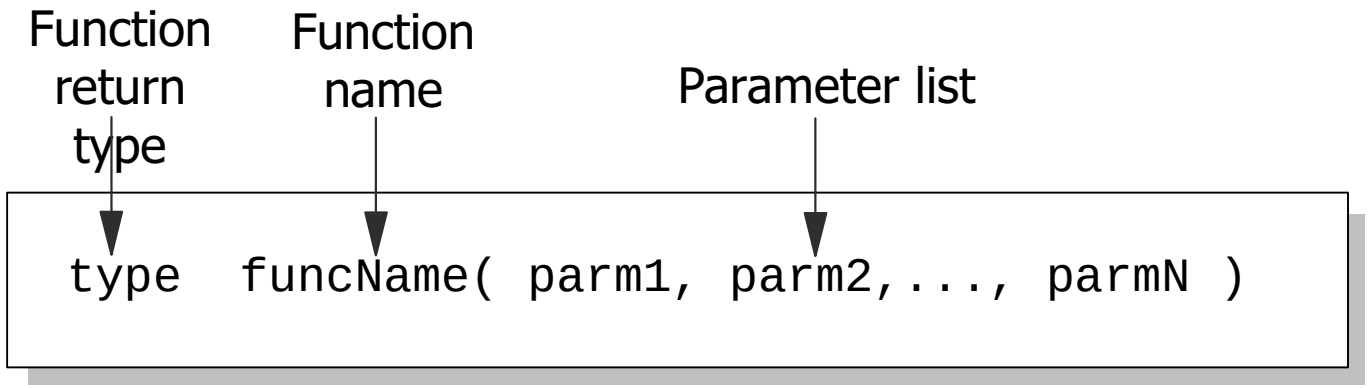
Function : Why use it

- Functions can hide details of operations
- Functions break large tasks into smaller ones.
- Structured programming
  - easy to write
  - easy to debug
  - easy to maintain

Function : How it works



Function Header




```
int GetLine( char *cLine, int iMaxNumChar )

float SquareRoot( float fNum )

void PrintHeading( int iPageNumber )

float RandomNumber( void )
```

A Short C Program

```
/* Calculate the product of 2 numbers */
#include  <stdio.h>

int  a, b, c;
int product( int x, int y );

main()
{
  printf("Enter a number: ");
  scanf( "%d", &a );
  printf("Enter another number: ");
  scanf( "%d", &b );
  c = product( a, b );
  printf( "%d times %d = %d\n", a, b, c );
}
int product( int x, int y )
{
  return( x * y );
}
```

Function prototype

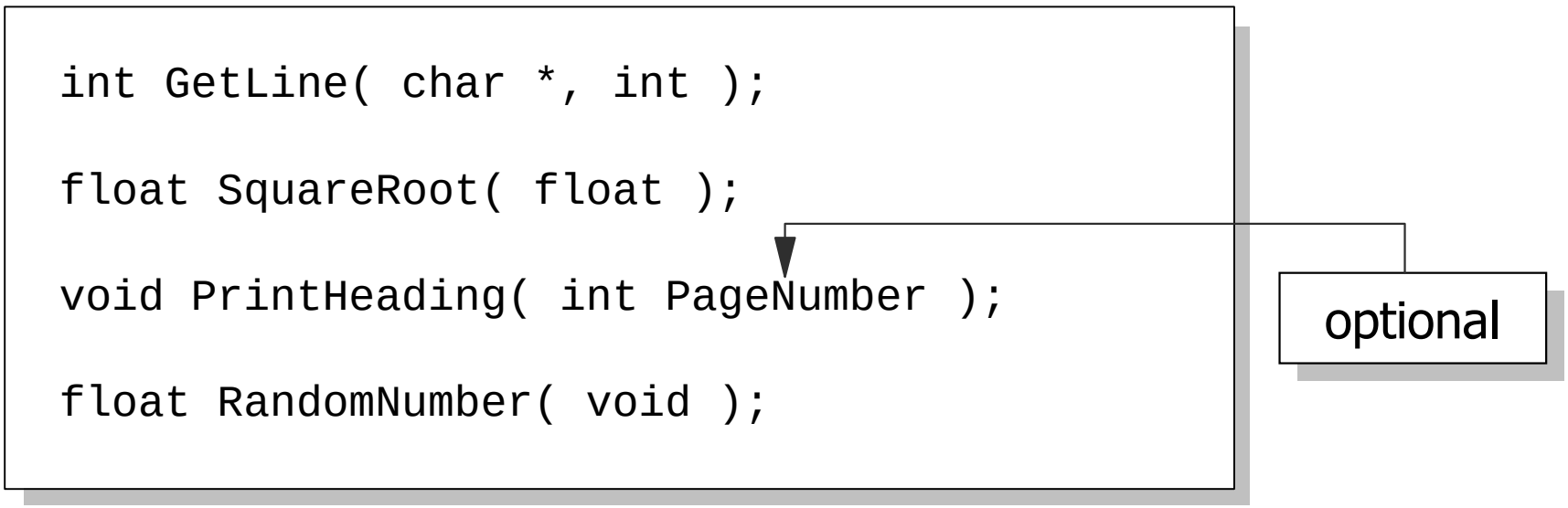
Function definition

Function definition

Function Prototype

- Provide the compiler with the description of a function that will be defined later.
- Always ends with a semicolon.

```
int GetLine( char *, int );

float SquareRoot( float );

void PrintHeading( int PageNumber );

float RandomNumber( void );
```



Function Definition

```
void reverse ( char s[] )
{
  int    c, i, j;

  for ( i=0,j=strlen(s)-1; i<j; i++,j++ ) {
    c    = s[i];
    s[i] = s[j];
    s[j] = c;
  }
}
```

```
function header
{
   function body
}
```

```
void dummy( void ) {}
```

Returning a Value

```
return  C_Expression ;
```

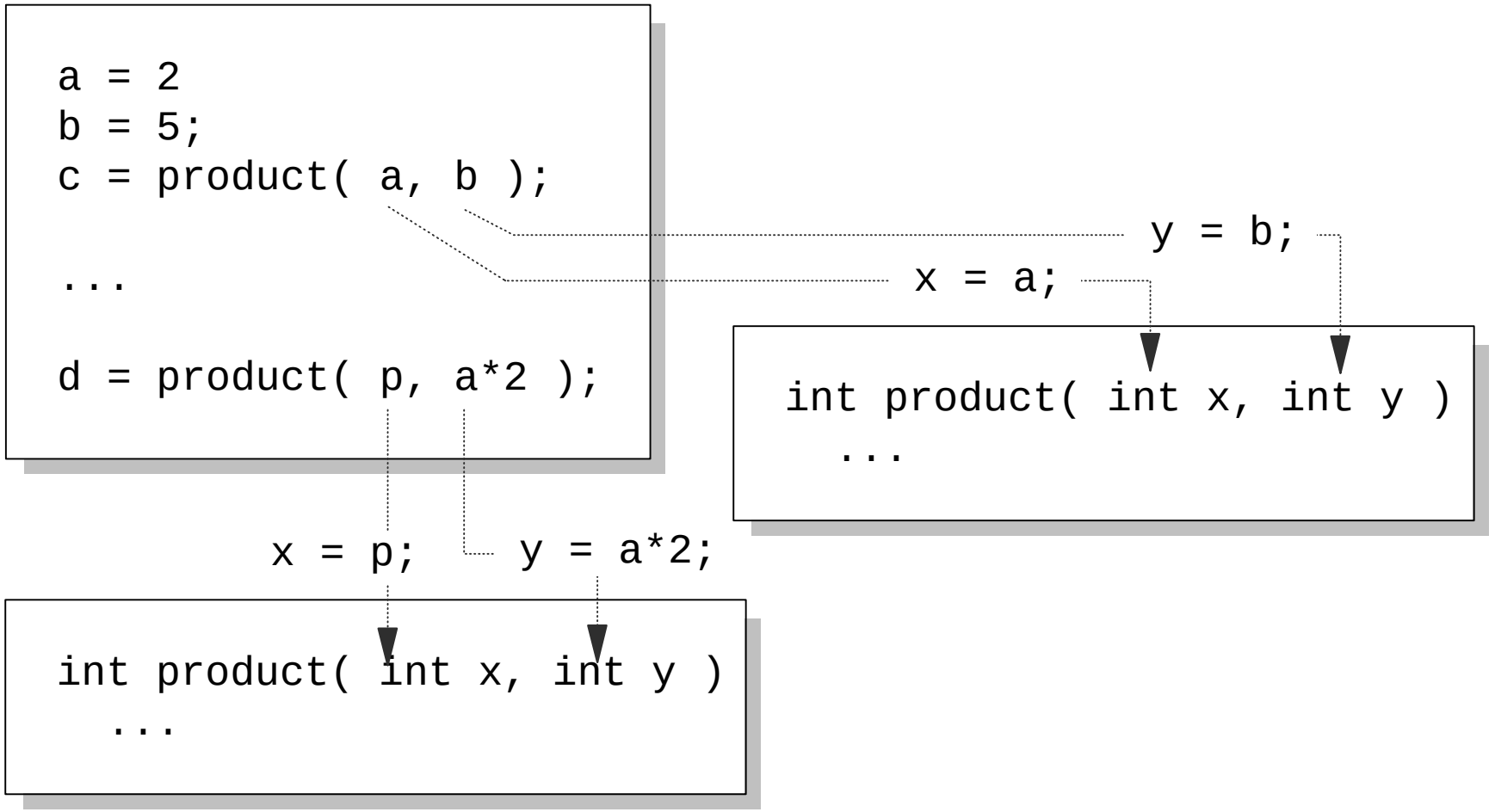
```
int LargerOf( int a, int b )
{
  if ( a > b )
    return a;
  else
    return b;
}
```

```
float Average( int n, float x[] )
{
  int      i;
  float    sum;

  for (sum=0,i=1; i<=n; i++)
    sum += x[i];
  return( sum / n );
}
```

Example

```
#include  <stdio.h>
int LargerOf ( int, int );
int GetNumber( void );

main()
{
  int     iMax, x;

  iMax = GetNumber();
  while ( (x = GetNumber()) > 0 )
    iMax = LargerOf( iMax, x );
  printf( "\nThe maximum value is %d\n", iMax );
}

int GetNumber( void )
{
  int     a;
  printf( "\nEnter a positive number (0 to quit) : " );
  scanf( "%d", &a );
  return a;
}
```

Passing Arguments



Passing Arguments

```
z = half(third(square(half(y))));
```

```
t1 = half(y);
t2 = square( t1 );
t3 = third( t2 );
z  = half( t3 );
```

Do & Don't

- Do use a function name that describes the purpose of the function.
- Do pass parameters to functions in order to make the function generic and reusable.
- Don't pass values to a function that it doesn't need.
- Don't pass fewer or more arguments to a function than there are parameters.
- Don't return a value that has a different type than the function's type.
- Don't let functions get too long.

Storage Classes

- Local variables (internal)
- Global variables (external)
- Automatic vs. Static variables
- Register variables

Local Variables

- Variables declared within the body function are private to that particular function.

```
main()
{
  int     iMax, x;

  iMax = GetNumber();
  while ( (x = GetNumber()) > 0 )
    iMax = (iMax > x) ? iMax : x;
  printf( "\nThe maximum value is %d\n", iMax );
}
int GetNumber( void )
{
  int     a;
  printf( "\nEnter a positive number (0 to quit) : " );
  scanf( "%d", &a );
  return a;
}
```

Local Variables

```
main()
{
  int     iMax, x;

  iMax = GetNumber();
  while ( (x = GetNumber()) > 0 )
    iMax = (iMax > x) ? iMax : x;
  printf( "\nThe maximum value is %d\n", iMax );
}
int GetNumber( void )
{
  int     x;
  printf( "\nEnter a positive number (0 to quit) : " );
  scanf( "%d", &x );
  return x;
}
```

Global Varaibles

```
main()
{
  int     iMax, x;

  iMax = GetNumber();
  while ( (x = GetNumber()) > 0 )
    iMax = (iMax > x) ? iMax : x;
  printf( "\nThe maximum value is %d\n", iMax );
}
int GetNumber( void )
{
  int     x;
  printf( "\nEnter a positive number (0 to quit) : " );
  scanf( "%d", &x );
  return x;
}
```

Global Variables

```
int     x;
main()
{
  int     iMax;

  iMax = GetNumber();
  while ( (x = GetNumber()) > 0 )
    iMax = (iMax > x) ? iMax : x;
  printf( "\nThe maximum value is %d\n", iMax );
}
int GetNumber( void )
{
  printf( "\nEnter a positive number (0 to quit) : " );
  scanf( "%d", &x );
  return x;
}
```

External variable

Global Variables

```
int     x;
main()
{
  int     iMax;

  GetNumber(); iMax = x;
  GetNumber();
  while ( x > 0 ) {
    iMax = (iMax > x) ? iMax : x;
    GetNumber();
  }
  printf( "\nThe maximum value is %d\n", iMax );
}
void GetNumber( void )
{
  printf( "\nEnter a positive number (0 to quit) : " );
  scanf( "%d", &x );
}
```

Static Variables

Automatic

```
int NextNumber()
{
  int     iCurrentNum = 0;

  iCurrentNum++;
  return( iCurrentNum );
}
```

Static

```
int NextNumber()
{
  static int  iCurrentNum = 0;

  iCurrentNum++;
  return( iCurrentNum );
}
```

Source Code File

maxnum.c

```
#include  <stdio.h>

int GetNumber( void );

main()
{
  int     iMax, x;

  iMax = GetNumber();
  while ( (x = GetNumber()) > 0 )
    iMax = (iMax > x) ? iMax : x;
  printf( "\nThe maximum value is %d\n", iMax );
}

int GetNumber( void )
{
  int     x;
  printf( "\nEnter a positive number (0 to quit) : " );
  scanf( "%d", &x );
  return x;
}
```

Source Code Files : extern

maxnum.c

```
#include  <stdio.h>

extern int GetNumber( void );

main()
{
  int     iMax, x;

  iMax = GetNumber();
  while ( (x = GetNumber()) > 0 )
    iMax = (iMax > x) ? iMax : x;
  printf( "\nThe maximum value is %d\n", iMax );
}
```

util.c

```
int GetNumber( void )
{
  int     x;
  printf( "\nEnter a positive number (0 to quit) : " );
  scanf( "%d", &x );
  return x;
}
```

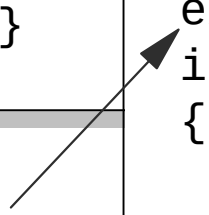
Source Code Files : `extern`

maxnum.c

```
#include  <stdio.h>

extern int GetNumber( void );
int    x;

main()
{
  int     iMax;

  iMax = GetNumber();
  while ( (x = GetNumber()) > 0 )
    iMax = (iMax > x) ? iMax : x;
  pri
}
```

util.c

```
extern int  x;
int GetNumber( void )
{
  printf( "\nEnter a positive number (0 to quit) : " );
  scanf( "%d", &x );
  return x;
}
```



Source Code Files : `extern`

maxnum.c

```
#include  <stdio.h>

extern int GetNumber( void );
int    x;

main()
{
  int     iMax;

  iMax = GetNumber();
  while ( (x = GetNumber()) > 0 )
    iMax = (iMax > x) ? iMax : x;
  pri
}
```

util.c

```
int GetNumber( void )
{
  extern int  x;
  printf( "\nEnter a positive number (0 to quit) : " );
  scanf( "%d", &x );
  return x;
}
```

Source Code Files : `extern`

maxnum.c

```
#include  <stdio.h>

extern int GetNumber( void );
extern int x;

main()
{
  int     iMax;

  iMax = GetNumber();
  while ( (x = GetNumber()) > 0 )
    iMax = (iMax > x) ? iMax : x;
  pri
}
```

util.c

```
int  x;
int GetNumber( void )
{
  printf( "\nEnter a positive number (0 to quit) : " );
  scanf( "%d", &x );
  return x;
}
```

Source Code Files : `static`

maxnum.c

```
#include  <stdio.h>

extern int GetNumber( void );
extern int x;

main()
{
  int     iMax;

  iMax = GetNumber();
  while ( (x = GetNumber()) > 0 )
    iMax = (iMax > x) ? iMax : x;
  pri
}
```

util.c

```
static int  x;
static int GetNumber( void )
{
  printf( "\nEnter a positive number (0 to quit) : " );
  scanf( "%d", &x );
  return x;
}
```

Register Variables

- A register declaration advises the compiler that the variable will be heavily used.
- If possible, a register variable will be placed in a very high speed memory.
- Only applied to automatic variables and parameters.
- Machine dependent.

```
int PostAnalysis( register int iNo )
{
  register int   iMin;
  register char  cPrevChar;
  double         dStdDeviation;
  ...
}
```

Scope : Summary

```
int   iGlobal1, iGlobal2;

void main( void )
{
  int      iAutoLocal;
  static int  iNotNecessary;
  ...
}
void PrintHeading( void )
{
  static int iStaticLocal =
  ...
}
```

```
static double  dStaticGlobal = 0;

void Analyse( void )
{
  extern   int    iGlobal1;
  register int    rLocalRegister;
  ...
}
int PreProcessing( int iLocalPara )
{
  ...
}
```

Do & Don't

- Do initialize local variables.
- Do initialize global variables.
- Do pass local variables as function parameters.
- Do put variable definitions at the beginning of the function
- Don't declare static variables in `main()`.

Which Class Should You Use ?

- Give each variable automatic local to begin with.
- Use register, if it is frequently used.
- Use static, if its value must be retained between calls.
- Use external storage class, if it is used by most or all of the program's functions.

Q & A

- If global variables can be used anywhere in the program, why not make all variables global ?
- True or false : a register variable always be placed in register ?
- Is it possible to do this ?

```
int    iNum;

void main( void )
{
  int      iNum;

  ...
}
```

Summation : Iteration

```
sum(1,...,n) = 1+2+3+,...,+n
```

```
int sum( int n )
{
  int    i, tmp = 0;

  for (i=1; i<=n; i++) tmp += i;
  return( tmp );
}
```

Recursion

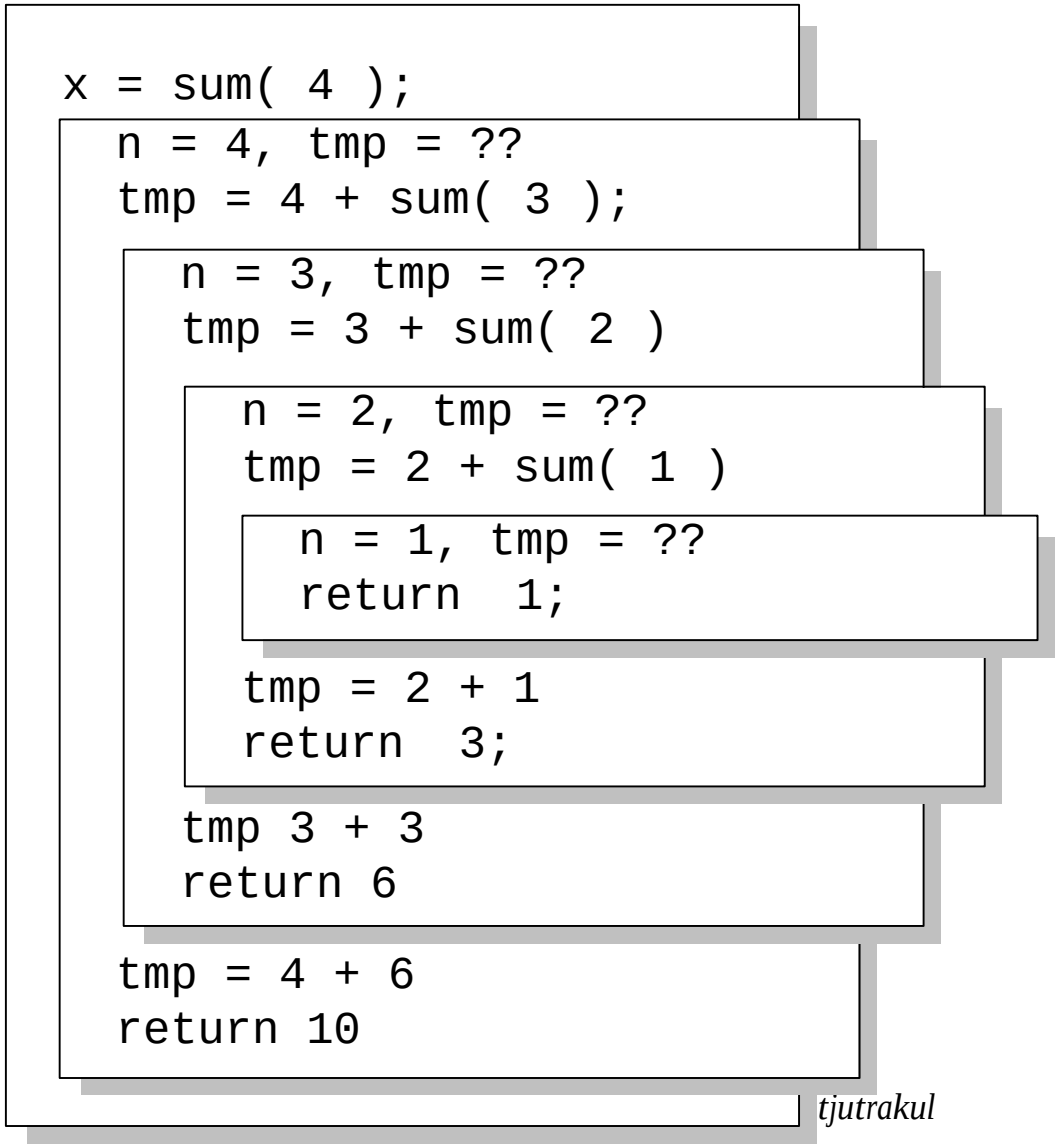
```
sum(1,...,n) = 1+2+3+,...,+n

sum(1,...,n)   = sum(1,...,n-1) + n
sum(1,...,n-1) = sum(1,...,n-2) + n
sum(1,...,n-2) = sum(1,...,n-3) + n
                 ...
sum(1,...,3)   = sum(1,...,2) + 3
sum(1,...,2)   = sum(1,...,1) + 2
sum(1,...,1)   = 1
```

```
int sum( int n )
{
  int    tmp;

  if ( n == 1 ) return 1;
  tmp = n + sum( n-1 );
  return( tmp );
}
```

Recursion

```
int sum( int n )
{
  int    tmp;

  if ( n == 1 ) return 1;
  tmp = n + sum( n-1 );
  return( tmp );
}
```

Whenever a function is called, its local variables and function parameters are newly created.



Recursion

```
sum(1,...,n) = 1+2+3+,...,+n

sum(1,...,n) = sum(1,...,n/2) + sum(1+n/2,...,n)
sum(a,...,b) = sum( a,...,(a+b)/2 ) +
               sum( 1+(a+b)/2,...,b )
sum(a,...,a) = a

sum(1,...,5) = sum(1,...,3) + sum(4,...,5)
sum(1,...,3) = sum(1,...,2) + sum(3,...,3)
sum(1,...,2) = sum(1,...,1) + sum(2,...,2)
sum(4,...,5) = sum(4,...,4) + sum(5,...,5)
```

Recursion

```
sum(1,...,n) = 1+2+3+,...,+n

sum(1,...,n) = sum(1,...,n/2) + sum(1+n/2,...,n)
sum(a,...,b) = sum( a,...,(a+b)/2 ) +
               sum( 1+(a+b)/2,...,b )
sum(a,...,a) = a

sum(1,...,5) = sum(1,...,3) + sum(4,...,5)
sum(1,...,3) = sum(1,...,2) + sum(3,...,3)
sum(1,...,2) = sum(1,...,1) + sum(2,...,2)
sum(4,...,5) = sum(4,...,4) + sum(5,...,5)
```

Recursion

```
sum(1,...,n) = 1+2+3+,...,+n

sum(1,...,n) = sum(1,...,n/2) + sum(1+n/2,...,n)
sum(a,...,b) = sum( a,...,(a+b)/2 ) +
               sum( 1+(a+b)/2,...,b )
sum(a,...,a) = a
```

```
int sum( int left, int right )
{
  int    mid, tmp;

  if ( left == right ) return left;
  mid = (left + right) / 2;
  tmp = sum(left,mid) + sum(mid+1,right);
  return tmp ;
}
```