C programming language

College of Saint Benedict & Saint John’s University

origins

Dennis Ritchie in 2011 / CC BY 2.0

Brian Kernighan in 2012 / CC BY 2.0

hello, world

```c
/* file: helloworld.c */

#include <stdio.h>

int main() {
  printf("hello, world\n");
  return 0;
}
```

```
$ gcc -o helloworld helloworld.c
$ ./helloworld
hello, world
```

global variables

```c
// file: figure2-4.c
// Stan Warford
// A nonsense program to illustrate global variables

#include <stdio.h>

char ch;
int j;

int main() {
  scanf("%c %d", &ch, &j);
  j += 5;
  ch++;
  printf("%c\n%d\n", ch, j);
  return 0;
}
```

```
$ gcc -o figure2-4 figure2-4.c
$ ./figure2-4
M 419
N
424
```

program breakdown

```c
#include <stdio.h>

char ch;
int j;

int main() {
  scanf("%c %d", &ch, &j);
  j += 5;
  ch++;
  printf("%c\n%d\n", ch, j);
  return 0;
}
```

C programs ALWAYS start execution with the `main` function

returning from `main` ends the program

program breakdown

global variables are declared here — outside of any function

characters in C are treated internally like signed integers

```c
#include <stdio.h>

char ch;
int j;

int main() {
  scanf("%c %d", &ch, &j);
  j += 5;
  ch++;
  printf("%c\n%d\n", ch, j);
  return 0;
}
```

program breakdown

```c
#include <stdio.h>

char ch;
int j;

int main() {
  scanf("%c %d", &ch, &j);
  j += 5;
  ch++;
  printf("%c\n%d\n", ch, j);
  return 0;
}
```

correct headers must be included to access library functions

read data from `stdin` (the terminal)

print data to `stdout` (the terminal)

`scanf` and `printf` are both library functions declared in `stdio.h`

program breakdown

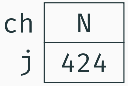
```
#include <stdio.h>

char ch;
int j;

int main() {
  scanf("%c %d", &ch, &j);
  j += 5;
  ch++;
  printf("%c\n%d\n", ch, j);
  return 0;
}
```

& is the address of operator — `scanf` expects the address of the variables where the data will be stored

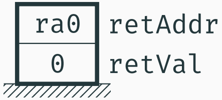
memory model — part i

global variables
: declared outside of any function and remain in place throughout the execution of the entire program. they are stored at a fixed location in memory.

local variables
: declared within a function and come into existence when the function is called and cease to exist when the function terminates. they are stored on the run-time stack.



(a) Fixed location.



(b) Run-time stack.

run-time stack a.k.a. “the stack”

run-time stack

stores information about the active functions of a C program, including:

- return value,
- actual parameters,
- return address, and
- local variables

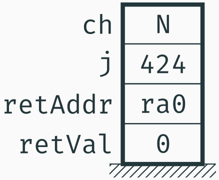
in that order.

run-time stack a.k.a. “the stack”

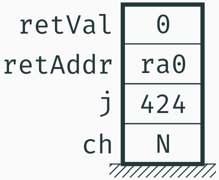
run-time stack
: stores information about the active functions of a C program, including:

- return value,
- actual parameters,
- return address, and
- local variables

in that order.

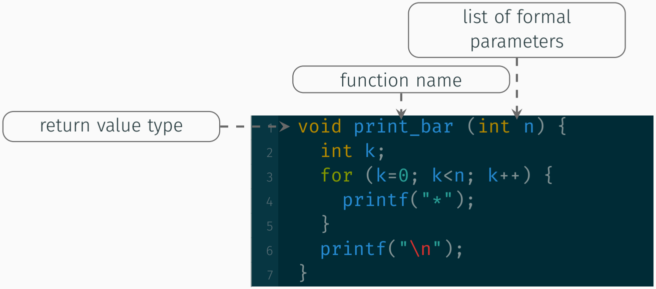


(a)



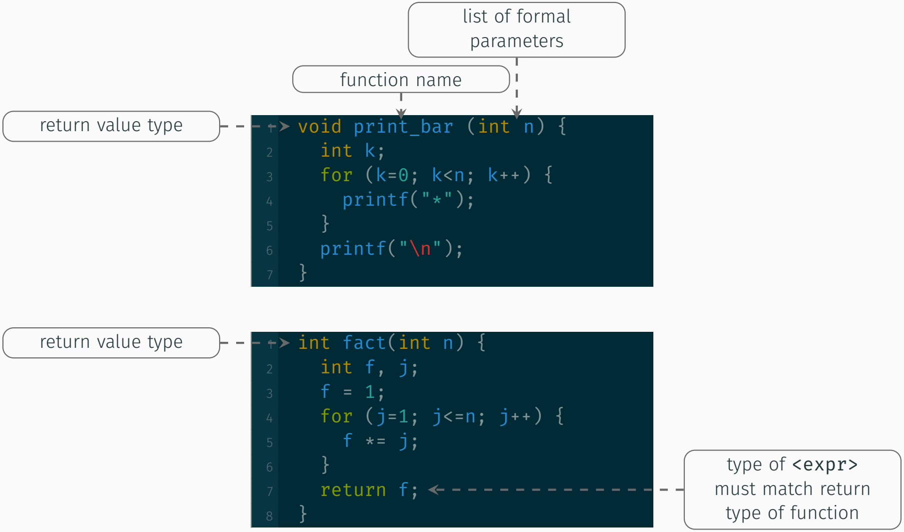
(b)

functions



```c
void print_bar (int n) {
  int k;
  for (k=0; k<n; k++) {
    printf("*");
  }
  printf("\n");
}
```

functions

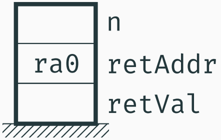

list of formal parameters
function name
return value type
return value type
type of <expr> must match return type of function

```c
void print_bar (int n) {
  int k;
  for (k=0; k<n; k++) {
    printf("*");
  }
  printf("\n");
}
```

```c
int fact(int n) {
  int f, j;
  f = 1;
  for (j=1; j<=n; j++) {
    f *= j;
  }
  return f;
}
```

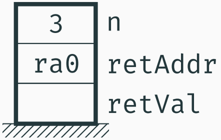
functions — call-by-value

```c
int fact(int n) {
  int f, j;
  f = 1;
  for (j=1; j<=n; j++) {
    f *= j;
  }
  return f;
}

int main() {
  int n;
  scanf("%d", &n);
  printf("%d\n", fact(n)); // ra1
  return 0;
}
```



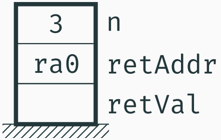
functions — call-by-value

```c
int fact(int n) {
  int f, j;
  f = 1;
  for (j=1; j<=n; j++) {
    f *= j;
  }
  return f;
}

int main() {
  int n;
  scanf("%d", &n);
  printf("%d\n", fact(n)); // ra1
  return 0;
}
```



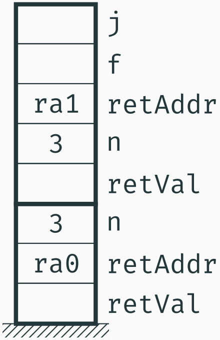
functions — call-by-value

```c
int fact(int n) {
  int f, j;
  f = 1;
  for (j=1; j<=n; j++) {
    f *= j;
  }
  return f;
}

int main() {
  int n;
  scanf("%d", &n);
  printf("%d\n", fact(n)); // ra1
  return 0;
}
```



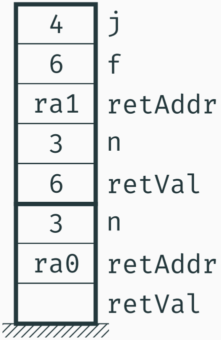
functions — call-by-value

```c
int fact(int n) {
  int f, j;
  f = 1;
  for (j=1; j<=n; j++) {
    f *= j;
  }
  return f;
}

int main() {
  int n;
  scanf("%d", &n);
  printf("%d\n", fact(n)); // ra1
  return 0;
}
```



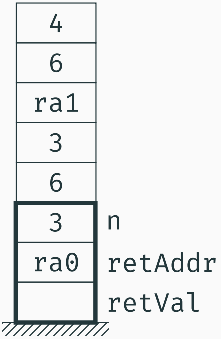
functions — call-by-value

```c
int fact(int n) {
  int f, j;
  f = 1;
  for (j=1; j<=n; j++) {
    f *= j;
  }
  return f;
}

int main() {
  int n;
  scanf("%d", &n);
  printf("%d\n", fact(n)); // ra1
  return 0;
}
```



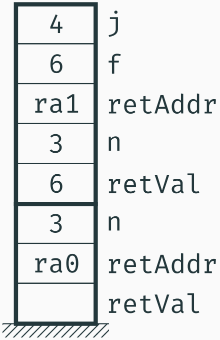
functions — call-by-value

```c
int fact(int n) {
  int f, j;
  f = 1;
  for (j=1; j<=n; j++) {
    f *= j;
  }
  return f;
}

int main() {
  int n;
  scanf("%d", &n);
  printf("%d\n", fact(n)); // ra1
  return 0;
}
```



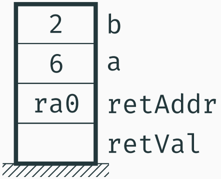
functions — call-by-value

```c
int fact(int n) {
  int f, j;
  // f = 1;
  for (j=1; j<=n; j++) {
    f *= j;
  }
  return f;
}

int main() {
  int n;
  scanf("%d", &n);
  printf("%d\n", fact(n));
  scanf("%d", &n);
  printf("%d\n", fact(n)); // ra1
  return 0;
}
```



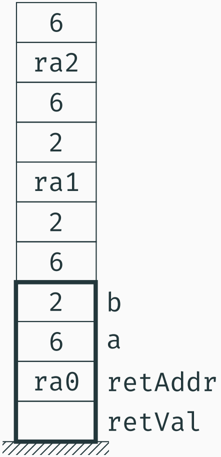
functions — call-by-“reference”

```c
void swap(int r, int s) {
  int temp;
  temp = r;
  r = s;
  s = temp;
}

void order(int x, int y) {
  if (x > y) {
    swap(x, y);
  } // ra2
}

int main() {
  int a, b;
  scanf("%d %d", &a, &b);
  order(a, b);
  printf("d %d\n", a, b); // ra1
  return 0;
}
```



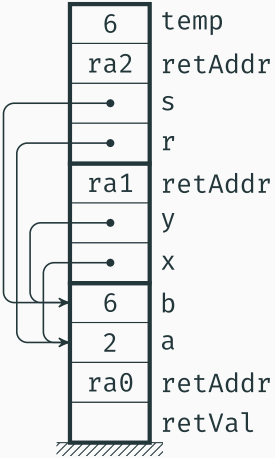
functions — call-by-“reference”

```c
void swap(int r, int s) {
  int temp;
  temp = r;
  r = s;
  s = temp;
}

void order(int x, int y) {
  if (x > y) {
    swap(x, y);
  } // ra2
}

int main() {
  int a, b;
  scanf("%d %d", &a, &b);
  order(a, b);
  printf("d %d\n", a, b); // ra1
  return 0;
}
```



functions — call-by-“reference”

```c
void swap(int *r, int *s) {
  int temp;
  temp = *r;
  *r = *s;
  *s = temp;
}

void order(int *x, int *y) {
  if (*x > *y) {
    swap(x, y);
  } // ra2
}

int main() {
  int a, b;
  scanf("%d %d", &a, &b);
  order(&a, &b);
  printf("d %d\n", a, b); // ra1
  return 0;
}
```

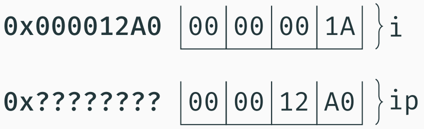


pointers

- a pointer is a variable whose value is a memory address

```c
int  i   = 0x1A;
int *ip = &i;
```

- `&i` evaluates to the address where the variable `i` is stored in memory
- `i` is an `int`, so `ip` is a *pointer* to an `int`



pointers cont.

```c
printf("0x%X\n", i);    /* 0x1A */
printf("0x%#X\n", &i);  /* 0x12A0 */
printf("0x%#X\n", ip);  /* 0x12A0 */
printf("0x%#X\n", &ip); /* 0x???????? */
```

pointer dereference

- `*ptr` will
  1. treat the value of `ptr` as a memory address
  2. get the bytes of data located at that memory address
  3. interpret those bytes according to the type of pointer that `ptr` is

```
printf("0x%X\n", *ip);    /* 0x1A */
```

pointer dereference

- `*ptr` will
  1. treat the value of `ptr` as a memory address
  2. get the bytes of data located at that memory address
  3. interpret those bytes according to the type of pointer that `ptr` is

```c
printf("0x%X\n", *ip);   /* 0x1A */
```

- `ip[X] = *(ip + X)`

```c
printf("0x%X\n", ip[0]); /* 0x1A */
```

pointers cont.

```
printf("0x%X\n", i);          /* 0x1A */
printf("0x%X\n", *ip);        /* 0x1A */
printf("0x%X\n", ip[0]);      /* 0x1A */
printf("0x%X\n", *(ip+0));    /* 0x1A */
printf("0x%#X\n", &i);        /* 0x12A0 */
printf("0x%#X\n", ip);        /* 0x12A0 */
printf("0x%#X\n", &ip);       /* 0x???????? */
```

pointers cont.

```c
char *cp = "hello, world";
```

- `cp` is a *pointer* to a `char`

0x00004C80	h	e	l	l	o	,		w	o	r	l	d	\0

0x????????	00	00	4C	80

```c
printf("%c\n", *cp);     /* h */
printf("%c\n", cp[0]);   /* h */
printf("%c\n", cp[4]);   /* o */
printf("%c\n", *(cp+4)); /* o */
printf("%s\n", cp);      /* hello, world */
printf("%s\n", cp+7);    /* world */
printf("0x%#X\n", cp);   /* 0x4C80 */
printf("0x%#X\n", &cp);  /* 0x???????? */
```

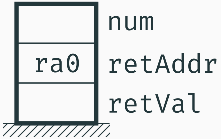
recursion

```c
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra1
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```

recursion

```
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra2
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```

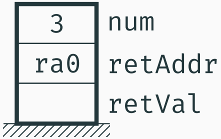
```
$ ./figure2-22
```



recursion

```c
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra2
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```

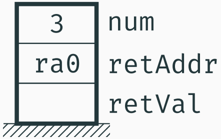
```
$ ./figure2-22
Enter a small integer: 3
```



recursion

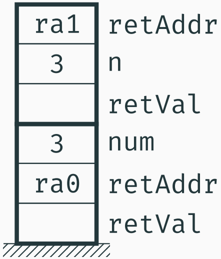
```c
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra2
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```

```
$ ./figure2-22
Enter a small integer: 3
```



recursion

```c
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra2
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```

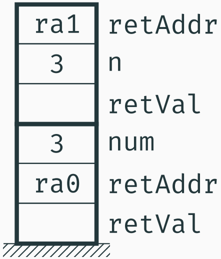


```
$ ./figure2-22
Enter a small integer: 3
```

recursion

```c
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra2
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```

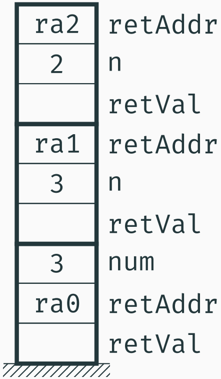
```
$ ./figure2-22
Enter a small integer: 3
```



recursion

```c
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra2
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```

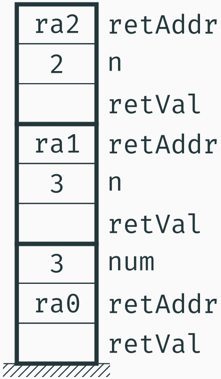
```
$ ./figure2-22
Enter a small integer: 3
```



recursion

```c
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra2
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```

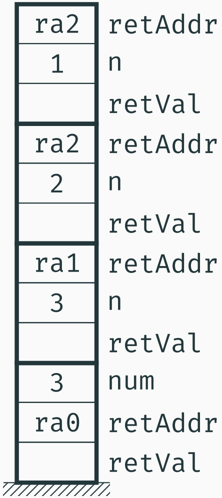
```
$ ./figure2-22
Enter a small integer: 3
```



recursion

```c
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra2
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```

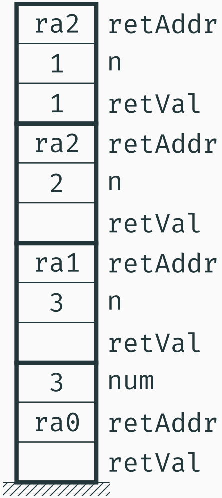
```
$ ./figure2-22
Enter a small integer: 3
```



recursion

```c
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra2
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```

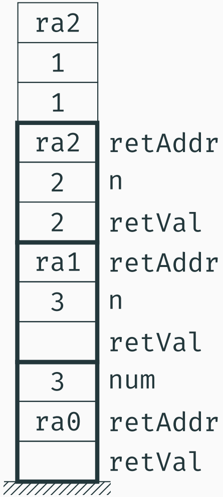
```
$ ./figure2-22
Enter a small integer: 3
```



recursion

```c
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra2
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```

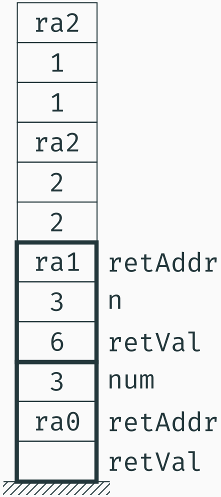
```
$ ./figure2-22
Enter a small integer: 3
```



recursion

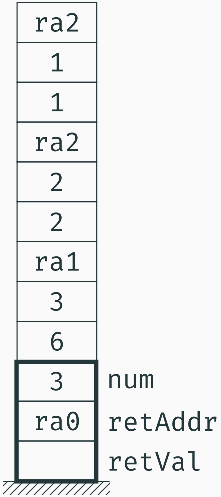
```c
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra2
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```

```
$ ./figure2-22
Enter a small integer: 3
```



recursion

```c
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra2
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```

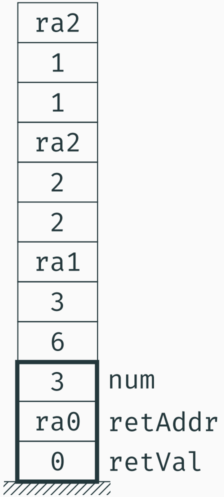


```
$ ./figure2-22
Enter a small integer: 3
Its factorial is: 6
```

recursion

```c
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra2
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```

```
$ ./figure2-22
Enter a small integer: 3
Its factorial is: 6
```



recursion

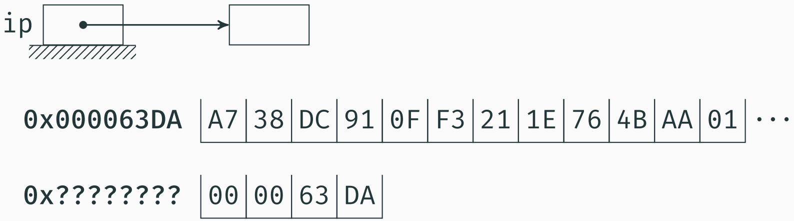
```c
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra2
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```

```
$ ./figure2-22
...
$
```

heap memory

- designate a block of memory to store value(s) of a particular data type

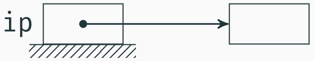
```c
int * ip = malloc(100*sizeof(int));
```

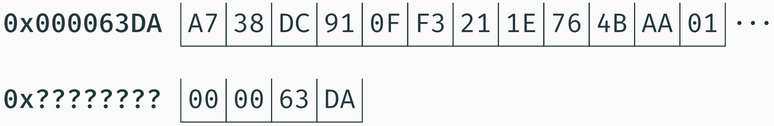


heap memory

- designate a block of memory to store value(s) of a particular data type

```
int * ip = malloc(100*sizeof(int));
```



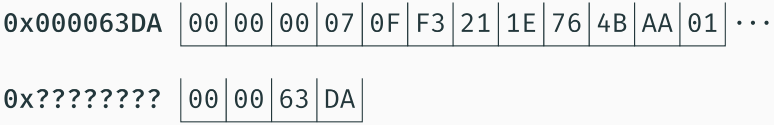


- release a block of memory back to system to be used elsewhere

```
free(ip);
```

heap memory cont.

```c
ip[0] = 0x7; /* *ip = 0x7; */
```



heap memory cont.

```c
ip[0] = 0x7; /* *ip = 0x7; */
```

```
0x000063DA  |00|00|00|07|0F|F3|21|1E|76|4B|AA|01| ···

0x????????  |00|00|63|DA|
```

```c
ip[1] = 0xA; /* *(ip + 1) = 0xA; */
```

```
0x000063DA  |00|00|00|07|00|00|00|0A|76|4B|AA|01| ···
```

structures

```c
#include <stdio.h>

int main() {
  struct {
    char first;
    char last;
    int age;
    char gender;
  } bill;

  scanf("%c%c%d%c", &bill.first, &bill.last, &bill.age,
    &bill.gender);
  printf("Initials: %c%c\n", bill.first, bill.last);
  printf("Age: %d\n", bill.age);
  printf("Gender: ");
  if (bill.gender == 'f') {
    printf("fe");
  }
  printf("male\n");
  return 0;
}
```

structures cont.

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

struct { char *name; } bill;

char * find(char *str, char c) {
  for (; *str != c; str++);
  return str;
}

int main() {
  char *first, *last;

  first = malloc(100);
  scanf("%s", first);
  last = find(first, '-') + 1;
  last[-1] = '\0';
  printf("Initials: %c%c\n", first[0], *last);
  free(first);
  bill.name = malloc(100);
  printf("Full name: %s\n", bill.name);
  free(bill.name);

  return 0;
}
```

comparison

Java	C
object-oriented	procedural
interpreted	compiled
`String`	`char` array
condition (`boolean`)	condition (`int`)
garbage-collected	no memory management
references	pointers
exceptions	error codes