ECE 264 Spring 2023

Advanced C Programming

Define New Types

```
// vector.h
#ifndef VECTOR_H
#define VECTOR_H
typedef struct
{
   int x;
   int y;
   int z;
} Vector; /* don't forget ; */
#endif
```

different
data types

```
// vector.h
#ifndef VECTOR_H
#define VECTOR_H
typedef struct
{
   int x;
   int y;
   int z;
   double t;
   char name[30];
} Vector; /* don't forget ; */
#endif
```

Why to create new data type?

- Organize information better
- Distinguish data types (abstract) from instances (“objects”)
- Reduce chances of mistakes
- Simplify data passing among functions
- Improve data consistency
- (in Object-Oriented Languages) protect class data from accidental changes

```c
// vector.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "vector.h"
int main(int argc, char * argv[])
{
  Vector v1;
  v1.x = 3;
  v1.y = 6;
  v1.z = -2;
  printf("The vector is (%d, %d, %d).\n",
         v1.x, v1.y, v1.z);
  return EXIT_SUCCESS;
}
```

```c
// vector.h
#ifndef VECTOR_H
#define VECTOR_H
typedef struct
{
  int x;
  int y;
  int z;
} Vector; /* don't forget ; */
#endif
```

Symbol	Address	Value
v1.z	308	-2
v1.y	304	6
v1.x	300	3

& v1.y = & v1.x + sizeof(v1.x)

& v1.z = & v1.x + sizeof(v1.x) + sizeof(v1.y)

```c
// vector2.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "vector.h"
int main(int argc, char * argv[])
{
  Vector v1;
  v1.x = 3;
  v1.y = 6;
  v1.z = -2;
  printf("The vector is (%d, %d, %d).\n",
         v1.x, v1.y, v1.z);
  Vector v2 = {0};
  printf("The vector is (%d, %d, %d).\n",
         v2.x, v2.y, v2.z);
  v2 = v1;
  printf("The vector is (%d, %d, %d).\n",
         v2.x, v2.y, v2.z);
  v1.x = -4;
  v2.y = 5;
  printf("The vector is (%d, %d, %d).\n",
         v1.x, v1.y, v1.z);
  printf("The vector is (%d, %d, %d).\n",
         v2.x, v2.y, v2.z);
  return EXIT_SUCCESS;
}
```

```
The vector is (3, 6, -2).
The vector is (0, 0, 0).
The vector is (3, 6, -2).
The vector is (-4, 6, -2).
The vector is (3, 5, -2).
```

```c
// vector2.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "vector.h"
int main(int argc, char * argv[])
{
  Vector v1;
  v1.x = 3;
  v1.y = 6;
  v1.z = -2;
  printf("The vector is (%d, %d, %d).\n",
         v1.x, v1.y, v1.z);
  Vector v2 = {0};
  printf("The vector is (%d, %d, %d).\n",
         v2.x, v2.y, v2.z);
  v2 = v1;
  printf("The vector is (%d, %d, %d).\n",
         v2.x, v2.y, v2.z);
  v1.x = -4;
  v2.y = 5;
  printf("The vector is (%d, %d, %d).\n",
         v1.x, v1.y, v1.z);
  printf("The vector is (%d, %d, %d).\n",
         v2.x, v2.y, v2.z);
  return EXIT_SUCCESS;
}
```

```
The vector is (3, 6, -2).
The vector is (0, 0, 0).
The vector is (3, 6, -2).
The vector is (-4, 6, -2).
The vector is (3, 5, -2).
```

Initialize all elements to zero

```c
// vector2.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "vector.h"
int main(int argc, char * argv[])
{
  Vector v1;
  v1.x = 3;
  v1.y = 6;
  v1.z = -2;
  printf("The vector is (%d, %d, %d).\n",
         v1.x, v1.y, v1.z);
  Vector v2 = {0};
  printf("The vector is (%d, %d, %d).\n",
         v2.x, v2.y, v2.z);
  v2 = v1;
  printf("The vector is (%d, %d, %d).\n",
         v2.x, v2.y, v2.z);
  v1.x = -4;
  v2.y = 5;
  printf("The vector is (%d, %d, %d).\n",
         v1.x, v1.y, v1.z);
  printf("The vector is (%d, %d, %d).\n",
         v2.x, v2.y, v2.z);
  return EXIT_SUCCESS;
}
```

```
The vector is (3, 6, -2).
The vector is (0, 0, 0).
The vector is (3, 6, -2).
The vector is (-4, 6, -2).
The vector is (3, 5, -2).
```

copy the attributes from v1 to v2

= (assignment) is the only supported operator

not supported: !=, <, <=, >, >=, ++, --

```c
// vector2.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "vector.h"
int main(int argc, char * argv[])
{
  Vector v1;
  v1.x = 3;
  v1.y = 6;
  v1.z = -2;
  printf("The vector is (%d, %d, %d).\n",
         v1.x, v1.y, v1.z);
  Vector v2 = {0};
  printf("The vector is (%d, %d, %d).\n",
         v2.x, v2.y, v2.z);
  v2 = v1;
  printf("The vector is (%d, %d, %d).\n",
         v2.x, v2.y, v2.z);
  v1.x = -4;
  v2.y = 5;
  printf("The vector is (%d, %d, %d).\n",
         v1.x, v1.y, v1.z);
  printf("The vector is (%d, %d, %d).\n",
         v2.x, v2.y, v2.z);
  return EXIT_SUCCESS;
}
```

```
The vector is (3, 6, -2).
The vector is (0, 0, 0).
The vector is (3, 6, -2).
The vector is (-4, 6, -2).
The vector is (3, 5, -2).
```

changing v1.x does not change v2.x
changing v2.y does not change v1.y

```c
// vector2.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "vector.h"
int main(int argc, char * argv[])
{
  Vector v1;
  v1.x = 3;
  v1.y = 6;
  v1.z = -2;
  printf("The vector is (%d, %d, %d).\n",
         v1.x, v1.y, v1.z);
  Vector v2 = {0};
  printf("The vector is (%d, %d, %d).\n",
         v2.x, v2.y, v2.z);
  v2 = v1;
  printf("The vector is (%d, %d, %d).\n",
         v2.x, v2.y, v2.z);
  v1.x = -4;
  v2.y = 5;
  printf("The vector is (%d, %d, %d).\n",
         v1.x, v1.y, v1.z);
  printf("The vector is (%d, %d, %d).\n",
         v2.x, v2.y, v2.z);
  return EXIT_SUCCESS;
}
```

Symbol	Address	Value
v1.z	308	U
v1.y	304	U
v1.x	300	U

```c
// vector2.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "vector.h"
int main(int argc, char * argv[])
{
  Vector v1;
  v1.x = 3;
  v1.y = 6;
  v1.z = -2;
  printf("The vector is (%d, %d, %d).\n",
         v1.x, v1.y, v1.z);
  Vector v2 = {0};
  printf("The vector is (%d, %d, %d).\n",
         v2.x, v2.y, v2.z);
  v2 = v1;
  printf("The vector is (%d, %d, %d).\n",
         v2.x, v2.y, v2.z);
  v1.x = -4;
  v2.y = 5;
  printf("The vector is (%d, %d, %d).\n",
         v1.x, v1.y, v1.z);
  printf("The vector is (%d, %d, %d).\n",
         v2.x, v2.y, v2.z);
  return EXIT_SUCCESS;
}
```

Symbol	Address	Value
v1.z	308	-2
v1.y	304	6
v1.x	300	3

```c
// vector2.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "vector.h"
int main(int argc, char * argv[])
{
  Vector v1;
  v1.x = 3;
  v1.y = 6;
  v1.z = -2;
  printf("The vector is (%d, %d, %d).\n",
         v1.x, v1.y, v1.z);
  Vector v2 = {0};
  printf("The vector is (%d, %d, %d).\n",
         v2.x, v2.y, v2.z);
  v2 = v1;
  printf("The vector is (%d, %d, %d).\n",
         v2.x, v2.y, v2.z);
  v1.x = -4;
  v2.y = 5;
  printf("The vector is (%d, %d, %d).\n",
         v1.x, v1.y, v1.z);
  printf("The vector is (%d, %d, %d).\n",
         v2.x, v2.y, v2.z);
  return EXIT_SUCCESS;
}
```

Symbol	Address	Value
v2.z	320	0
v2.y	316	0
v2.x	312	0
v1.z	308	-2
v1.y	304	6
v1.x	300	3

```c
// vector2.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "vector.h"
int main(int argc, char * argv[])
{
  Vector v1;
  v1.x = 3;
  v1.y = 6;
  v1.z = -2;
  printf("The vector is (%d, %d, %d).\n",
         v1.x, v1.y, v1.z);
  Vector v2 = {0};
  printf("The vector is (%d, %d, %d).\n",
         v2.x, v2.y, v2.z);
  v2 = v1;
  printf("The vector is (%d, %d, %d).\n",
         v2.x, v2.y, v2.z);
  v1.x = -4;
  v2.y = 5;
  printf("The vector is (%d, %d, %d).\n",
         v1.x, v1.y, v1.z);
  printf("The vector is (%d, %d, %d).\n",
         v2.x, v2.y, v2.z);
  return EXIT_SUCCESS;
}
```

```
The vector is (3, 6, -2).
The vector is (0, 0, 0).
The vector is (3, 6, -2).
The vector is (-4, 6, -2).
The vector is (3, 5, -2).
```

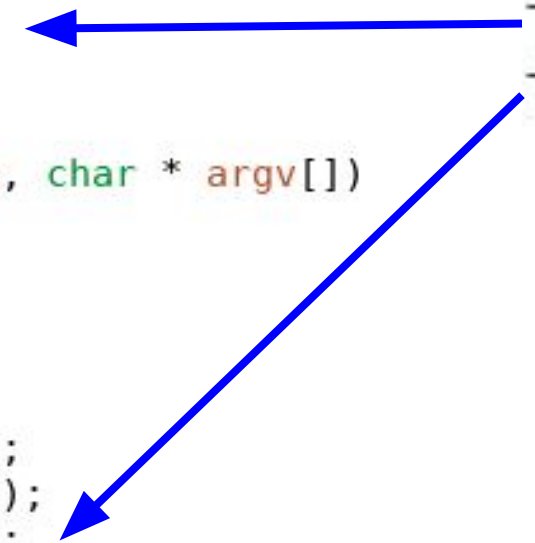
Symbol	Address	Value
v2.z	320	-2
v2.y	316	6
v2.x	312	3
v1.z	308	-2
v1.y	304	6
v1.x	300	3

```c
// vector4.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "vector.h"
void printVector(Vector v)
{
  printf("The vector is (%d, %d, %d).\n", v.x, v.y, v.z);
}

void changeVector(Vector v)
{
  v.x = 5;
  v.y = -3;
  v.z = 7;
  printVector(v);
}

int main(int argc, char * argv[])
{
  Vector v1;
  v1.x = 3;
  v1.y = 6;
  v1.z = -2;
  printVector(v1);
  changeVector(v1);
  printVector(v1);
  return EXIT_SUCCESS;
}
```

```
The vector is (3, 6, -2).
The vector is (5, -3, 7).
The vector is (3, 6, -2).
```

```c
// vector4.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "vector.h"
void printVector(Vector v)
{
  printf("The vector is (%d, %d, %d).\n", v.x, v.y, v.z);
}

void changeVector(Vector v)
{
  v.x = 5;
  v.y = -3;
  v.z = 7;
  printVector(v);
}

int main(int argc, char * argv[])
{
  Vector v1;
  v1.x = 3;
  v1.y = 6;
  v1.z = -2;
  printVector(v1);
  changeVector(v1);
  printVector(v1);
  return EXIT_SUCCESS;
}
```

```
The vector is (3, 6, -2).
The vector is (5, -3, 7).
The vector is (3, 6, -2).
```



Vector did not change in main (remember local variables)

```c
// vector4.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "vector.h"
void printVector(Vector v)
{
  printf("The vector is (%d, %d, %d).\n", v.x, v.y, v.z);
}

void changeVector(Vector v)
{
  v.x = 5;
  v.y = -3;
  v.z = 7;
  printVector(v);
}

int main(int argc, char * argv[])
{
  Vector v1;
  v1.x = 3;
  v1.y = 6;
  v1.z = -2;
  printVector(v1);
  changeVector(v1);
  printVector(v1);
  return EXIT_SUCCESS;
}
```

Frame	Symbol	Address	Value
changeVector	v.z	320	-2
	v.y	316	6
	v.x	312	3
main	v1.z	308	-2
	v1.y	304	6
	v1.x	300	3

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "vector.h"
void printVector(Vector v)
{
  printf("The vector is (%d, %d, %d).\n", v.x, v.y, v.z);
}

void changeVector(Vector * v)
{
  v -> x = 5;
  v -> y = -3;
  v -> z = 7;
  printVector(* v);
}

int main(int argc, char * argv[])
{
  Vector v1;
  v1.x = 3;
  v1.y = 6;
  v1.z = -2;
  printVector(v1);
  changeVector(& v1);
  printVector(v1);
  return EXIT_SUCCESS;
}
```

```
The vector is (3, 6, -2).
The vector is (5, -3, 7).
The vector is (5. -3. 7).
```

Passing structure by pointer

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "vector.h"
void printVector(Vector v)
{
  printf("The vector is (%d, %d, %d).\n", v.x, v.y, v.z);
}

void changeVector(Vector * v)
{
  v -> x = 5;
  v -> y = -3;
  v -> z = 7;
  printVector(* v);
}

int main(int argc, char * argv[])
{
  Vector v1;
  v1.x = 3;
  v1.y = 6;
  v1.z = -2;
  printVector(v1);
  changeVector(& v1);
  printVector(v1);
  return EXIT_SUCCESS;
}
```

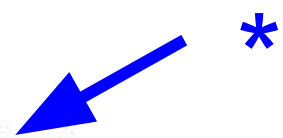
```
The vector is (3, 6, -2).
The vector is (5, -3, 7).
The vector is (5. -3. 7).
```

&

Frame	Symbol	Address	Value
changeVector	v	320	A300
main	v1.z	308	-2
	v1.y	304	6
	v1.x	300	3

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "vector.h"
void printVector(Vector v)
{
  printf("The vector is (%d, %d, %d).\n", v.x, v.y, v.z);
}

void changeVector(Vector * v)
{
  v -> x = 5;
  v -> y = -3;
  v -> z = 7;
  printVector(* v);
}

int main(int argc, char * argv[])
{
  Vector v1;
  v1.x = 3;
  v1.y = 6;
  v1.z = -2;
  printVector(v1);
  changeVector(& v1);
  printVector(v1);
  return EXIT_SUCCESS;
}
```

```
The vector is (3, 6, -2).
The vector is (5, -3, 7).
The vector is (5. -3. 7).
```



right hand side rule

Frame	Symbol	Address	Value
changeVector	v	320	A300
main	v1.z	308	-2
	v1.y	304	6
	v1.x	300	3

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "vector.h"
void printVector(Vector v)
{
  printf("The vector is (%d, %d, %d).\n", v.x, v.y, v.z);
}

void changeVector(Vector * v)
{
  Vector v2 = {.x = 5, .y = 7, .z = 9};
  * v = v2;
  printVector(* v);
}

int main(int argc, char * argv[])
{
  Vector v1;
  v1.x = 3;
  v1.y = 6;
  v1.z = -2;
  printVector(v1);
  changeVector(& v1);
  printVector(v1);
  return EXIT_SUCCESS;
}
```

another way to assign values

left hand side rule

right hand side rule

```
The vector is (3, 6, -2).
The vector is (5, 7, 9).
The vector is (5, 7, 9).
```

Syntax . and ->

- If it is a pointer, use ->
- // right hand side and left hand side rules apply
- If it is not a pointer (called “object” in this class), use .

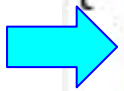
```
Vector v; // object, not a pointer
v.x = 264;
Vector * vp = & v;
vp -> y = 2020;
```

Struct can be use in another struct

```c
// dateofbirth.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct
{
  int year;
  int month;
  int date;
} DateOfBirth;

typedef struct
{
  char * name;
  DateOfBirth dob;
} Person;
```

```c
// vector.h
#ifndef VECTOR_H
#define VECTOR_H
typedef struct
{
  int x;
  int y;
  int z;
} Vector; /* don't forget ; */
#endif
```

```c
#include "vector.h"
Vector * Vector_construct(int a, int b, int c)
// notice *
{
  Vector * v;
  v = malloc(sizeof(Vector));
  if (v == NULL) // allocation fail
    {
      printf("malloc fail\n");
      return NULL;
    }
  v -> x = a;
  v -> y = b;
  v -> z = c;
  return v;
}

void Vector_destruct(Vector * v)
{
  free (v);
}

void Vector_print(Vector * v)
{
  printf("The vector is (%d, %d, %d).\n",
         v -> x, v -> y, v -> z);
}
```



```c
int main(int argc, char * * argv)
{
  Vector * v1;
  v1 = Vector_construct(3, 6, -2);
  if (v1 == NULL)
    {
      return EXIT_FAILURE;
    }
  Vector_print(v1);
  Vector_destruct(v1);
  return EXIT_SUCCESS;
}
```

Frame	Symbol	Address	Value
main	v1	100	U

```c
int main(int argc, char * * argv)
{
  Vector * v1;
  v1 = Vector_construct(3, 6, -2);
  if (v1 == NULL)
    {
      return EXIT_FAILURE;
    }
  Vector_print(v1);
  Vector_destruct(v1);
  return EXIT_SUCCESS;
}
```

```c
#include "vector.h"
Vector * Vector_construct(int a, int b, int c)
// notice *
{
  Vector * v;
  v = malloc(sizeof(Vector));
  if (v == NULL) // allocation fail
    {
      printf("malloc fail\n");
      return NULL;
    }
  v -> x = a;
  v -> y = b;
  v -> z = c;
```

Frame	Symbol	Address	Value
Vector_construct	c	308	-2
	b	304	6
	a	300	3
	return location		
main	v1	100	U

```c
          v -> x, v -> y, v -> z);
}
```

```c
#include "vector.h"
Vector * Vector_construct(int a, int b, int c)
// notice *
{
  Vector * v;
  v = malloc(sizeof(Vector));
  if (v == NULL) // allocation fail
    {
      printf("malloc fail\n");
      return NULL;
    }
  v -> x = a;
  v -> y = b;
  v -> z = c;
  return v;
}

void Vector_destruct(Vector * v)
{
  free (v);
}
```

Frame	Symbol	Address	Value
Vector_construct	v	312	U
	c	308	-2
	b	304	6
	a	300	3
	return location		
main	v1	100	U

```c
#include "vector.h"
Vector * Vector_construct(int a, int b, int c)
// notice *
{
  Vector * v;
  v = malloc(sizeof(Vector));
  if (v == NULL) // allocation fail
    {
      printf("malloc fail\n");
      return NULL;
    }
  v -> x = a;
  v -> y = b;
  v -> z = c;
  return v;
}

void Vector_destruct(Vector * v)
{
  free (v);
}

void Vector_print(Vector * v)
```

Heap Memory

Symbol	Address	Value
v -> z	10008	U
v -> y	10004	U
v -> x	10000	U

Stack Memory

Frame	Symbol	Address	Value
Vector_construct	v	312	A10000
	c	308	-2
	b	304	6
	a	300	3
	return location		
main	v1	100	U

```c
#include "vector.h"
Vector * Vector_construct(int a, int b, int c)
// notice *
{
  Vector * v;
  v = malloc(sizeof(Vector));
  if (v == NULL) // allocation fail
    {
      printf("malloc fail\n");
      return NULL;
    }
  v -> x = a;
  v -> y = b;
  v -> z = c;
  return v;
}

void Vector_destruct(Vector * v)
{
  free (v);
}

void Vector_print(Vector * v)
```

Heap Memory		
Symbol	Address	Value
v -> z	10008	-2
v -> y	10004	6
v -> x	10000	3

Stack Memory			
Frame	Symbol	Address	Value
Vector_construct	v	312	A10000
	c	308	-2
	b	304	6
	a	300	3
	return location		
main	v1	100	U

```c
#include "vector.h"
Vector * Vector_construct(int a, int b, int c)
// notice *
{
  Vector * v;
  v = malloc(sizeof(Vector));
  if (v == NULL) // allocation fail
    {
      printf("malloc fail\n");
      return NULL;
    }
  v -> x = a;
  v -> y = b;
  v -> z = c;
  return v;
}

void Vector_destruct(Vector * v)
{
  free (v);
}

void Vector_print(Vector * v)
```

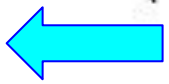
Heap Memory

Symbol	Address	Value
v -> z	10008	-2
v -> y	10004	6
v -> x	10000	3

Stack Memory

Frame	Symbol	Address	Value
Vector_construct	v	312	A10000
	c	308	-2
	b	304	6
	a	300	3
	return location		
main	v1	100	A10000

```c
int main(int argc, char * * argv)
{
  Vector * v1;
  v1 = Vector_construct(3, 6, -2);
  if (v1 == NULL)
    {
      return EXIT_FAILURE;
    }
  Vector_print(v1);
  Vector_destruct(v1);
  return EXIT_SUCCESS;
}
```



Heap Memory		
Symbol	Address	Value
v1 -> z	10008	-2
v1 -> y	10004	6
v1 -> x	10000	3

Frame	**Symbol**	**Address**	**Value**
main	v1	100	A10000

```c
int main(int argc, char * * argv)
{
  Vector * v1;
  v1 = Vector_construct(3, 6, -2);
  if (v1 == NULL)
    {
      return EXIT_FAILURE;
    }
  Vector_print(v1);
  Vector_destruct(v1);
  return EXIT_SUCCESS;
}
```

Heap Memory		
Symbol	Address	Value
v -> z	10008	-2
v -> y	10004	6
v -> x	10000	3

Frame	Symbol	Address	Value
main	v1	100	A10000

```c
int main(int argc, char * * argv)
{
  Vector * v1;
  v1 = Vector_construct(3, 6, -2);
  if (v1 == NULL)
    {
      return EXIT_FAILURE;
    }
  Vector_print(v1);
  Vector_destruct(v1);
  return EXIT_SUCCESS;
}
```

Heap Memory		
Symbol	Address	Value
~~v -> z~~	10008	-2
~~v -> y~~	10004	6
~~v -> x~~	10000	3

Frame	Symbol	Address	Value
main	v1	100	A10000

free does not change v1's value
v1's value is not NULL

```
Vector * p = NULL;
p = malloc(sizeof(Vector));
p -> x = 264; // ok
free (p);
if (p == NULL)  // will be false
…
p -> x = 2020; // segmentation fault
```

free does not change p's value
p's value is not NULL

```
Vector * p = NULL;
p = malloc(sizeof(Vector));
p -> x = 264; // ok
free (& p);
```

Will this set p's value to NULL?
No
p is a local variable on stack memory
& p is an address in stack memory

Segmentation Fault

- Computer memory is divided into units called segments.
- Each program is given some segments.
- Segmentation fault: a program intends to access (read from or write to) memory that does not belong to this program.
- Operating systems stop the program.
- To prevent segmentation fault:
  1. malloc before using
  2. do not use after free
  3. do not free twice

```c
// person.h
#ifndef PERSON_H
#define PERSON_H
typedef struct
{
  int year;
  int month;
  int date;
  char * name;
} Person;
```

```c
Person * Person_construct(char * n, int y, int m, int d)
{
  Person * p;
  p = malloc(sizeof(Person));
  if (p == NULL)
    {
      printf("malloc fail\n");
      return NULL;
    }
  p -> name = malloc(sizeof(char) * (strlen(n) + 1));
  /* + 1 for the ending character '\0' */
  strcpy(p -> name, n);
  p -> year = y;
  p -> month = m;
  p -> date = d;
  return p;
}
```

notice the order

```c
int main(int argc, char * argv[])
{
  Person * p1 = Person_construct("Amy", 1989, 8, 21);
  Person * p2 = Person_construct("Jennifer", 1991, 2, 17);
  Person * p3 = Person_copy(p1); // create p3
  Person_print(p1);
  Person_print(p2);
  Person_print(p3);
  p3 = Person_assign(p3, p2);
  Person_print(p3);
  Person_destruct(p1);
  Person_destruct(p2);
  Person_destruct(p3);
  return EXIT_SUCCESS;
}
```

```c
Person * Person_copy(Person * p)
{
  return Person_construct(p -> name, p -> year,
                          p -> month, p -> date);
}

Person * Person_assign(Person * p1, Person * p2)
{
  Person_destruct(p1);
  return Person_copy(p2);
}

void Person_print(Person * p)
{
  printf("Name: %s. ", p -> name);
  printf("Date of Birth: %d/%d/%d\n",
         p -> year, p -> month, p -> date);
}
```

```c
Person * Person_construct(char * n, int y, int m, int d)
{
  Person * p;
  p = malloc(sizeof(Person));
  if (p == NULL)
    {
      printf("malloc fail\n");
      return NULL;
    }
  p -> name = malloc(sizeof(char) * (strlen(n) + 1));
  /* + 1 for the ending character '\0' */
  strcpy(p -> name, n);
  p -> year = y;
  p -> month = m;
  p -> date = d;
  return p;
}

void Person_destruct(Person * p)
{
  free (p -> name);
  free (p);
}
```

malloc earlier will be free later

```c
Person * Person_construct(char * n, int y, int m, int d)
{
  Person * p;
  p = malloc(sizeof(Person));
  if (p == NULL)
    {
      printf("malloc fail\n");
      return NULL;
    }
  p -> name = malloc(sizeof(char) * (strlen(n) + 1));
  /* + 1 for the ending character '\0' */
  strcpy(p -> name, n);
  p -> year = y;
  p -> month = m;
  p -> date = d;
  return p;
}

void Person_destruct(Person * p)
{
  free (p -> name);
  free (p);
}
```

malloc p before
malloc p -> name

free p -> name before
free p

shallow vs deep copy

If a structure has one or several pointers, be very careful about assignment.

```
Person * p1 = Person_construct("Amy", 1989, 8, 21);
Person * p2 = Person_construct("Jennifer", 1991, 2, 17);
Person * p3 = Person_copy(p1); // create p3
Person * p4 = p1;
p3 = Person_assign(p3, p2); // change p3
// different from p3 = p2?
```

```
Person * p1 = Person_construct("Amy", 1989, 8, 21);
```

```
Person * Person_construct(char * n, int y, int m, int d)
{
  Person * p;
  p = malloc(sizeof(Person));
  if (p == NULL)
    {
      printf("malloc fail\n");
      return NULL;
    }
  p -> name = malloc(sizeof(char) * (strlen(n) + 1));
  /* + 1 for the ending character '\0' */
  strcpy(p -> name, n);
  p -> year = y;
  p -> month = m;
  p -> date = d;
  return p;
}
```

Frame	Symbol	Address	Value
main	p1	100	U

```
Person * p1 = Person_construct("Amy", 1989, 8, 21);
```

```
Person * Person_construct(char * n, int y, int m, int d)
{
  Person * p;
  p = malloc(sizeof(Person));
  if (p == NULL)
    {
      printf("malloc fail\n");
      return NULL;
    }
  p -> name = malloc(sizeof(char) * (strlen(n)
  /* + 1 for the ending character '\0' */
  strcpy(p -> name, n);
  p -> year = y;
  p -> month = m;
  p -> date = d;
  return p;
}
```

Frame	Symbol	Address	Value
construct	n[3]	223	\0
	n[2]	222	y
	n[1]	221	m
	n[0]	220	A
	d	216	21
	m	212	8
	y	208	1989
	n	200	A220
	value address		A100
	return location		
main	p1	100	U

```
Person * p1 = Person_construct("Amy", 1989, 8, 21);
```

```
Person * Person_construct(char * n, int y, int
{
  Person * p;
  p = malloc(sizeof(Person));
  if (p == NULL)
    {
      printf("malloc fail\n");
      return NULL;
    }
  p -> name = malloc(sizeof(char) * (strlen(n)
  /* + 1 for the ending character '\0' */
  strcpy(p -> name, n);
  p -> year = y;
  p -> month = m;
  p -> date = d;
  return p;
}
```

Frame	Symbol	Address	Value
construct	p	224	U
	n[3]	223	\0
	n[2]	222	y
	n[1]	221	m
	n[0]	220	A
	d	216	21
	m	212	8
	y	208	1989
	n	200	A220
	value address		A100
	return location		
main	p1	100	U

```
Person * p1 = Person_construct("Amy"
```

```
Person * Person_construct(char * n, int y, int
{
  Person * p;
  p = malloc(sizeof(Person));
  if (p == NULL)
    {
      printf("malloc fail\n");
      return NULL;
    }
  p -> name = malloc(sizeof(char) * (strlen(n)
  /* + 1 for the ending character '\0' */
  strcpy(p
  p -> yea
  p -> mon
  p -> dat
  return p
}
```

Heap Memory

Symbol	Address	Value
p->name	10012	
p->date	10008	
p->month	10004	
p->year	10000	

Stack Memory

Frame	Symbol	Address	Value
construct	p	224	A10000
	n[3]	223	\0
	n[2]	222	y
	n[1]	221	m
	n[0]	220	A
	d	216	21
	m	212	8
	y	208	1989
	n	200	A220
	value address		A100
	return location		
main	p1	100	U

```
Person * p1 = Person_construct("Amy", 1989, 8, 21);
```

```
Person * Person_construct(char * n, int y, int m, int d)
{
  Person * p;
  p = malloc(sizeof(Person));
  if (p == NULL)
    {
      printf("malloc fail\n");
      return NULL;
    }
  p -> name = malloc(sizeof(char) * (strlen(n) + 1));
  /* + 1 for the ending character '\0' */
  strcpy(p -> name, n);
  p -> year = y;
  p -> month = m;
  p -> date = d;
  return p;
}
```

Heap Memory		
Symbol	Address	Value
p->name[3]	25003	
p->name[2]	25002	
p->name[1]	25001	
p->name[0]	25000	
p->name	10012	A25000
p->date	10008	
p->month	10004	
p->year	10000	

```
Person * p1 = Person_construct("Amy"
```

```
Person * Person_construct
{
  Person * p;
  p = malloc(sizeof(Perso
  if (p == NULL)
    {
      printf("malloc fail
      return NULL;
    }
  p -> name = malloc(size
  /* + 1 for the ending c
  strcpy(p -> name, n);
  p -> year = y;
  p -> month = m;
  p -> date = d;
  return p;
}
```

Heap Memory

Symbol	Address	Value
p->name[3]	25003	\0
p->name[2]	25002	y
p->name[1]	25001	m
p->name[0]	25000	A
p->name	10012	A25000
p->date	10008	
p->month	10004	
p->year	10000	

Stack Memory

	Symbol	Address	Value
	p	224	A10000
	n[3]	223	\0
	n[2]	222	y
	n[1]	221	m
	n[0]	220	A
	d	216	21
	m	212	8
	y	208	1989
	n	200	A220
	value address		A100
	return location		
main	p1	100	U

```
Person * p1 = Person_construct("Amy"
```

```
Person * Person_construct
{
  Person * p;
  p = malloc(sizeof(Perso
  if (p == NULL)
    {
      printf("malloc fail
      return NULL;
    }
  p -> name = malloc(size
  /* + 1 for the ending c
  strcpy(p -> name, n);
  p -> year = y;
  p -> month = m;
  p -> date = d;
  return p;
}
```

Heap Memory

Symbol	Address	Value
p->name[3]	25003	\0
p->name[2]	25002	y
p->name[1]	25001	m
p->name[0]	25000	A
p->name	10012	A25000
p->date	10008	21
p->month	10004	8
p->year	10000	1989

Stack Memory

	Symbol	Address	Value
	p	224	A10000
	n[3]	223	\0
	n[2]	222	y
	n[1]	221	m
	n[0]	220	A
	d	216	21
	m	212	8
	y	208	1989
	n	200	A220
	value address		A100
	return location		
main	p1	100	U

```
Person * p1 = Person_construct("Amy"
```

```
Person * Person_construct
{
  Person * p;
  p = malloc(sizeof(Perso
  if (p == NULL)
    {
      printf("malloc fail
      return NULL;
    }
  p -> name = malloc(size
  /* + 1 for the ending c
  strcpy(p -> name, n);
  p -> year = y;
  p -> month = m;
  p -> date = d;
  return p;
}
```

Heap Memory

Symbol	Address	Value
p->name[3]	25003	\0
p->name[2]	25002	y
p->name[1]	25001	m
p->name[0]	25000	A
p->name	10012	A25000
p->date	10008	21
p->month	10004	8
p->year	10000	1989

Stack Memory

	Symbol	Address	Value
	p	224	A10000
	n[3]	223	\0
	n[2]	222	y
	n[1]	221	m
	n[0]	220	A
	d	216	21
	m	212	8
	y	208	1989
	n	200	A220
	value address		A100
	return location		
main	p1	100	A10000

```
Person * p1 = Person_construct("Amy",
```

```
Person * Person_construct(char * n, int y, int m
{
  Person * p;
  p = mallo
  if (p ==
    {
      print
      retur
    }
  p -> name
  /* + 1 fo
  strcpy(p
  p -> year
  p -> mont
  p -> date
  return p;
}
```

Heap Memory

Symbol	Address	Value
p->name[3]	25003	\0
p->name[2]	25002	y
p->name[1]	25001	m
p->name[0]	25000	A
p->name	10012	A25000
p->date	10008	21
p->month	10004	8
p->year	10000	1989

Stack Memory

Frame	Symbol	Address	Value
construct	p	224	A10000
	n[3]	223	\0
	n[2]	222	y
	n[1]	221	m
	n[0]	220	A
	d	216	21
	m	212	8
	y	208	1989
	n	200	A220
	value address		A100
	return location		
main	p1	100	A10000

```
Person * p3 = Person_copy(p1);
```

```
Person * Person_copy(Person * p)
{
  return Person_construct(p -> name, p -> year,
                          p -> month, p -> date);
}

Person * Pe                                    2)
{
  Person_de
  return Pe
}

void Person
{
  printf("N
  printf("D
          p
}
```

Heap Memory		
Symbol	Address	Value
p3->name[3]	45003	\0
p3->name[2]	45002	y
p3->name[1]	45001	m
p3->name[0]	45000	A
p3->name	20012	A45000
p3->date	20008	21
p3->month	20004	8
p3->year	20000	1989

Heap Memory		
Symbol	Address	Value
p1->name[3]	25003	\0
p1->name[2]	25002	y
p1->name[1]	25001	m
p1->name[0]	25000	A
p1->name	10012	A25000
p1->date	10008	21
p1->month	10004	8
p1->year	10000	1989

Stack Memory			
Frame	Symbol	Address	Value
main	p3	108	A20000
	p1	100	A10000

```
Person * p4 = p1;
```

Heap Memory		
Symbol	Address	Value
p->name[3]	25003	\0
p->name[2]	25002	y
p->name[1]	25001	m
p->name[0]	25000	A
p->name	10012	A25000
p->date	10008	21
p->month	10004	8
p->year	10000	1989

Stack Memory			
Frame	Symbol	Address	Value
main	p4	108	A10000
	p1	100	A10000

p1 and p4 point to the same heap memory
Changing p1 -> year changes p4 -> year

```
Person * p3 = Person_copy(p1);
Person * p4 = p1;
p3 = p4; // lose memory
```

Heap Memory		
Symbol	Address	Value
p3->name[3]	45003	\0
p3->name[2]	45002	y
p3->name[1]	45001	m
p3->name[0]	45000	A
p3->name	20012	A45000
p3->date	20008	21
p3->month	20004	8
p3->year	20000	1989

Heap Memory		
Symbol	Address	Value
p1->name[3]	25003	\0
p1->name[2]	25002	y
p1->name[1]	25001	m
p1->name[0]	25000	A
p1->name	10012	A25000
p1->date	10008	21
p1->month	10004	8
p1->year	10000	1989

Stack Memory			
Frame	Symbol	Address	Value
main	p4	116	A10000
	p3	108	A20000
	p1	100	A10000

```
Person * p3 = Person_copy(p1);
Person * p4 = p1;
p3 = p4; // lose memory
```

Heap Memory		
Symbol	Address	Value
p3->name[3]	45003	\0
p3->name[2]	45002	y
p3->name[1]	45001	m
p3->name[0]	45000	A
p3->name	20012	A45000
p3->date	20008	21
p3->month	20004	8
p3->year	20000	1989

lost

Heap Memory		
Symbol	Address	Value
p1->name[3]	25003	\0
p1->name[2]	25002	y
p1->name[1]	25001	m
p1->name[0]	25000	A
p1->name	10012	A25000
p1->date	10008	21
p1->month	10004	8
p1->year	10000	1989

Stack Memory			
Frame	Symbol	Address	Value
main	p4	116	A10000
	p3	108	A10000
	p1	100	A10000

```
Person p1; // no *
p1.year = 2001; // . not ->
p1.month = 3;
p1.date = 9;
p1.name = strdup("Amy");
Person p2 = p1;
```

```
typedef struct
{
  int year;
  int month;
  int date;
  char * name;
} Person;
```

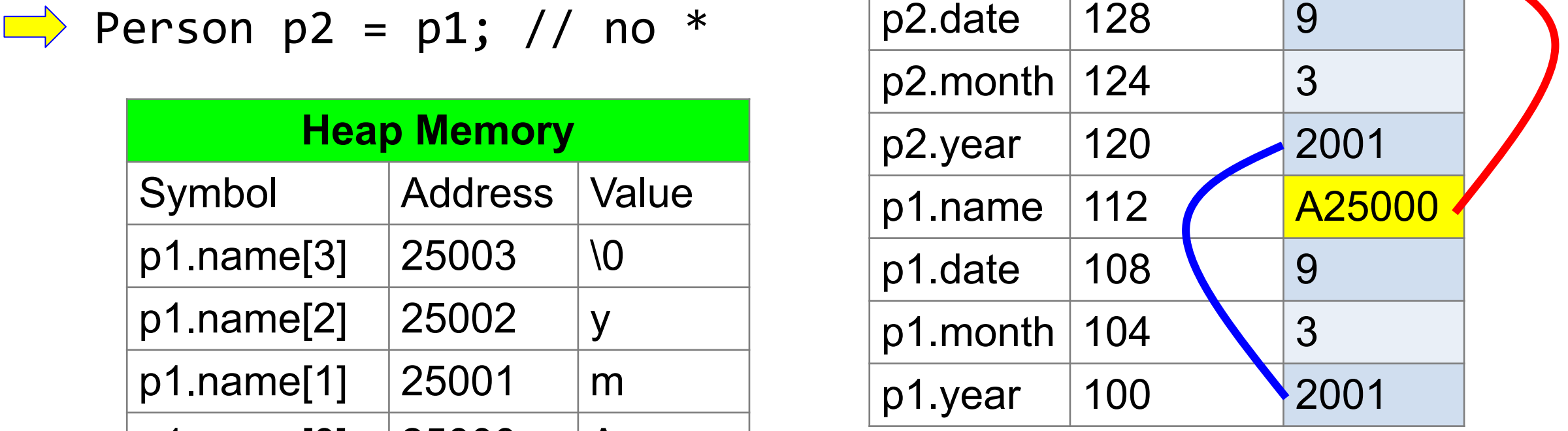
Heap Memory		
Symbol	Address	Value
p1.name[3]	25003	\0
p1.name[2]	25002	y
p1.name[1]	25001	m
p1.name[0]	25000	A

Stack Memory		
Symbol	Address	Value
p1.name	112	A25000
p1.date	108	9
p1.month	104	3
p1.year	100	2001

```
Person p1;
p1.year = 2001;
p1.month = 3;
p1.date = 9;
p1.name = strdup("Amy");
Person p2 = p1; // no *
```

Heap Memory		
Symbol	Address	Value
p1.name[3]	25003	\0
p1.name[2]	25002	y
p1.name[1]	25001	m
p1.name[0]	25000	A

Stack Memory		
Symbol	Address	Value
p2.name	132	A25000
p2.date	128	9
p2.month	124	3
p2.year	120	2001
p1.name	112	A25000
p1.date	108	9
p1.month	104	3
p1.year	100	2001



Shallow vs Deep Copy

- If a structure has one (or more) pointer, be very careful.
- Assignment (such as `p2 = p1;`) copies attribute by attribute.
- If an attribute is a pointer, two pointers refer to the same address (shallow copy).
- Shallow copy: changing p2.name[0] also changes p1.name[0]
- Deep copy: allocate memory so that they occupy different heap memory space

Shallow Copy	Deep Copy
point to the same heap memory	point to different heap memory
save memory space	use more memory
changing one changes the other (s)	changing one does not affect the other (s)
can be used when sharing is desired	can be used when sharing is not preferred
use case: address of employees	use case: address of children
Conclusion: Both are useful. You need to know which one to choose.	
“Copy-on-write”: beyond the scope of ECE 264.	

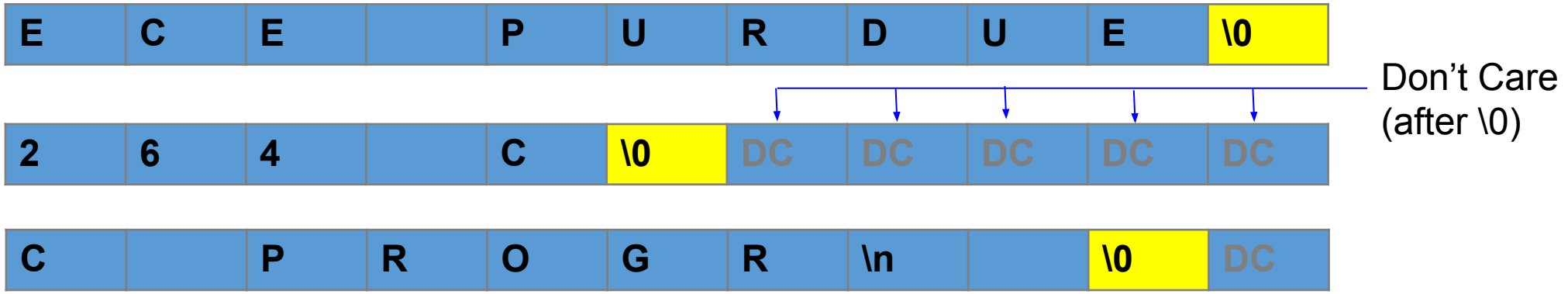
Homework 06
Count Occurrences of a Word

an article (in a file)
+ a word

count (a number)

Understand C Strings

- C has no “string” data type.
- C uses “array of characters + \0” as a string
- Each element can store a value between 0 and 255
- Conversion between numbers and characters based on ASCII



String Functions

```
#include <string.h>

size_t strlen(const char *s);
```

The **strlen**() function calculates the length of the string pointed to by *s*, excluding the terminating null byte ('\0').

```
char *strcpy(char *dest, const char *src);
```

The **strcpy**() function copies the string pointed to by *src*, including the terminating null byte ('\0'), to the buffer pointed to by *dest*. The strings may not overlap, and the destination string *dest* must be large enough to receive the copy. *Beware of buffer overruns!* (See BUGS.)

String Functions

The **strlen**() function calculates the length of the string pointed to by *s*, excluding the terminating null byte ('\0').

```
char *strcpy(char *dest, const char *src);
```

pay attention to the order

The **strcpy**() function copies the string pointed to by *src*, including the terminating null byte ('\0'), to the buffer pointed to by *dest*. The strings may not overlap, and the destination string *dest* must be large enough to receive the copy. *Beware of buffer overruns!* (See BUGS.)

C array is always a pointer

```
char * arr1;
arr1 = malloc(sizeof(char) * 20);
   // arr1 stores the address of the first element
strcpy(arr1, “Purdue ECE”);
char arr2[20];
   // arr2 is equivalent to & arr2[0];
   // i.e., address of the first element
   // cannot free (arr2)
free (arr1);
```

Double quotation automatically add ‘\0’

A pointer may not be an array

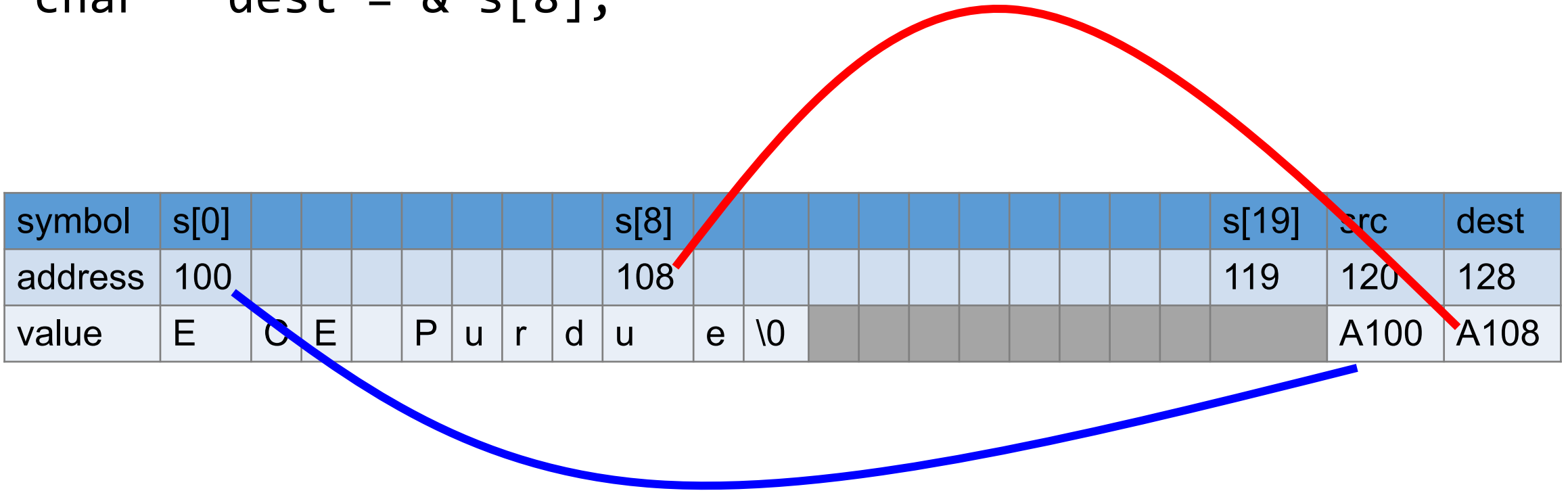
```
char ch = ‘A’;
char * p;
p = & ch; // a pointer, but there is no array
```

strcpy, not overlap

The **strcpy**() function copies the string pointed to by *src*, including the terminating null byte ('\0'), to the buffer pointed to by *dest*. The strings may not overlap, and the destination string *dest* must be large enough to receive the copy. *Beware of buffer overruns!* (See BUGS.)

```
char s[20];
strcpy(s, “ECE Purdue”);
char * src = & s[0];
char * dest = & s[8];
```

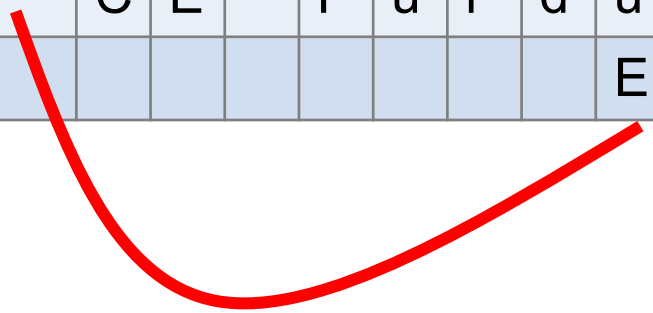
```
char s[20];
strcpy(s, “ECE Purdue”);
char * src = & s[0];
char * dest = & s[8];
```



symbol	s[0]								s[8]										s[19]	src	dest
address	100								108										119	120	128
value	E	C	E		P	u	r	d	u	e	\0									A100	A108

```
char s[20];
strcpy(s, “ECE Purdue”);
char * src = & s[0];
char * dest = & s[8];
strcpy(dest, src);
```

symbol	s[0]								s[8]										s[19]	src	dest
address	100								108										119	120	128
value	E	C	E		P	u	r	d	u	e	\0									A100	A108
									E	C	E										



String Functions

The **strlen**() function calculates the length of the string pointed to by *s*, excluding the terminating null byte ('\0').

```
char *strcpy(char *dest, const char *src);
```

The **strcpy**() function copies the string pointed to by *src*, including the terminating null byte ('\0'), to the buffer pointed to by *dest*. The strings may not overlap, and the destination string *dest* must be large enough to receive the copy. *Beware of buffer overruns!* (See BUGS.)

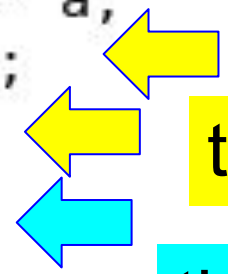
const in argument

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

void func(int * a, int * b)
{
  int t = * a;
  *a = * b;
  *b = t;
}

int main(int argc, char * * argv)
{
  int x = 123;
  int y = -456;
  func(&x, & y);
  printf("x = %d, y = %d\n", x, y);
  return EXIT_SUCCESS;
}
```

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

void func(int * a, const int * b)
{
  int t = * a;
  *a = * b;
  *b = t;
}

int main(int argc, char * * argv)
{
  int x = 123;
  int y = -456;
  func(&x, & y);
  printf("x = %d, y = %d\n", x, y);
  return EXIT_SUCCESS;
}
```

```
bash-4.2$ gcc const1.c
const1.c: In function ‘func’:
const1.c:8:3: error: assignment of read-only location ‘*b’
   *b = t;
   ^
```

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

void func(int * a, const int * b)
{
  int t = * a;
  *a = * b;
  b = a;
  *b = t;
}

int main(int argc, char * * argv)
{
  int x = 123;
  int y = -456;
  func(&x, & y);
  printf("x = %d, y = %d\n", x, y);
  return EXIT_SUCCESS;
}
```



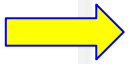
this is ok

this is not allowed (cannot use the LHS rule)

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

void func(int * a, const int * b)
{
  int t = * a;
  *a = * b;
  b = a;
  *b = t;
}

int main(int argc, char * * argv)
{
  int x = 123;
  int y = -456;
  func(&x, & y);
  printf("x = %d, y = %d\n", x, y);
  return EXIT_SUCCESS;
}
```

Frame	Symbol	Address	Value
func	t	212	123
	b	208	A104
	a	200	A100
main	y	104	-456
	x	100	123

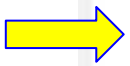
```c
#include <stdio.h>
#include <stdlib.h>

void func(int * a, const int * b)
{
  int t = * a;
  *a = * b;
  b = a;
  *b = t;
}

int main(int argc, char * * argv)
{
  int x = 123;
  int y = -456;
  func(&x, & y);
  printf("x = %d, y = %d\n", x, y);
  return EXIT_SUCCESS;
}
```



Frame	Symbol	Address	Value
func	t	212	123
	b	208	A104
	a	200	A100
main	y	104	-456
	x	100	~~123~~

-456

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

void func(int * a, const int * b)
{
  int t = * a;
  *a = * b;
  b = a;
  *b = t;
}

int main(int argc, char * * argv)
{
  int x = 123;
  int y = -456;
  func(&x, & y);
  printf("x = %d, y = %d\n", x, y);
  return EXIT_SUCCESS;
}
```



Frame	Symbol	Address	Value
func	t	212	123
	b	208	**A100**
	a	200	A100
main	y	104	-456
	x	100	-456

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

void func(const int * b)
{
  int * t = b;
  * t = 264;
}

int main(int argc, char * * argv)
{
  int y = -456;
  func(& y);
  printf("y = %d\n", y);
  return EXIT_SUCCESS;
}
```

```
bash-4.2$ gcc const2.c
const2.c: In function ‘func’:
const2.c:6:13: error: initialization discards ‘const’ qualifier from pointer target type [-Werror]
   int * t = b;
             ^
```

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

void func(const int * b)
{
  const int * t = b;
  * t = 264;
}

int main(int argc, char * * argv)
{
  int y = -456;
  func(& y);
  printf("y = %d\n", y);
  return EXIT_SUCCESS;
}
```

```
bash-4.2$ gcc const2.c
const2.c: In function ‘func’:
const2.c:7:3: error: assignment of read-only location ‘*t’
   * t = 264;
   ^
```

```
char *strdup(const char *s);
```

The *strdup*() function shall return a pointer to a new string, which is a duplicate of the string pointed to by *s*. The returned pointer can be passed to *free*(). A null pointer is returned if the new string cannot be created.

```
char *strstr(const char *haystack, const char *needle);
```

The **strstr**() function finds the first occurrence of the substring *needle* in the string *haystack*. The terminating null bytes ('\0') are not compared.

char *strdup(const char *s);

The *strdup*() function shall return a pointer to a new string, which is a duplicate of the string pointed to by *s*. The returned pointer can be passed to *free*(). A null pointer is returned if the new string cannot be created.

char *strstr(const char *haystack, const char *needle);

The **strstr**() function finds the first occurrence of the substring *needle* in the string *haystack*. The terminating null bytes ('\0') are not compared.

‘\0’ in string

- The array must have space to store this special character
- strelen does not count it

```
char * mystrdup(const char * src)
{
   char * p = malloc(sizeof(char) * (strlen(src) + 1));
   strcpy(p, src);
   return p;
}
```

without + 1,
program behavior undefined

for ‘\0’

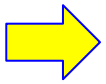
strstr

```
char *t = “PCE ECECECECE”;
// How many ECE does t have?
// Does “ECECE” count as one or two?
char * p;
p = strstr(t, “ECE”);
```

Symbol	Address	Value
p	222	U
t	214	A200
	213	\0
	212	E
	211	C
	210	E
	209	C
	208	E
	207	C
	206	E
	205	C
	204	E
	203	
t[2]	202	E
t[1]	201	C
t[0]	200	P

strstr

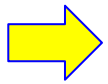
```
char *t = “PCE ECECECECE”;
// How many ECE does t have?
// Does “ECECE” count as one or two?
char * p;
p = strstr(t, “ECE”);
```



Symbol	Address	Value
p	222	A204
t	214	A200
	213	\0
	212	E
	211	C
	210	E
	209	C
	208	E
	207	C
	206	E
	205	C
	204	E
	203	
t[2]	202	E
t[1]	201	C
t[0]	200	P

strstr

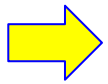
```
char *t = “PCE ECECECECE”;
// How many ECE does t have?
// Does “ECECE” count as one or two?
char * p;
p = strstr(t, “ECE”);
p = strstr(t, “ECE”); // p is 204
```



Symbol	Address	Value
p	222	A204
t	214	A200
	213	\0
	212	E
	211	C
	210	E
	209	C
	208	E
	207	C
	206	E
	205	C
	204	E
	203	
t[2]	202	E
t[1]	201	C
t[0]	200	P

strstr

```
char *t = “PCE ECECECECE”;
// How many ECE does t have?
// Does “ECECE” count as one or two?
char * p;
p = strstr(t, “ECE”);
p = strstr(p, “ECE”); // p is 204
```



Symbol	Address	Value
p	222	A204
t	214	A200
	213	\0
	212	E
	211	C
	210	E
	209	C
	208	E
	207	C
	206	E
	205	C
	204	E
	203	
t[2]	202	E
t[1]	201	C
t[0]	200	P

strstr

```
char *t = “PCE ECECECECE”;
// How many ECE does t have?
// Does “ECECE” count as one or two?
char * p;
p = strstr(t, “ECE”);
p ++;
```

Symbol	Address	Value
p	222	A205
t	214	A200
	213	\0
	212	E
	211	C
	210	E
	209	C
	208	E
	207	C
	206	E
	205	C
	204	E
	203	
t[2]	202	E
t[1]	201	C
t[0]	200	P

strstr

```
char *t = “PCE ECECECECE”;
// How many ECE does t have?
// Does “ECECE” count as one or two?
char * p;
p = strstr(t, “ECE”);
p ++;
char * q = p;
```

Symbol	Address	Value
q	230	A205
p	222	A206
t	214	A200
	213	\0
	212	E
	211	C
	210	E
	209	C
	208	E
	207	C
	206	E
	205	C
	204	E
	203	
t[2]	202	E
t[1]	201	C
t[0]	200	P

strstr

```
char *t = “PCE ECECECECE”;
// How many ECE does t have?
// Does “ECECE” count as one or two?
char * p;
p = strstr(t, “ECE”);
p ++;
char * q = p;
p = strstr(q, “ECE”); // not t
```

Symbol	Address	Value
q	230	A205
p	222	A206
t	214	A200
	213	\0
	212	E
	211	C
	210	E
	209	C
	208	E
	207	C
	206	E
	205	C
	204	E
	203	
t[2]	202	E
t[1]	201	C
t[0]	200	P

```
char *t = “PCE ECECECECE”;
// How many ECE does t have?
// Does “ECECE” count as one or two?
char * p;
p = strstr(t, “ECE”);
p ++;
char * q = p;
p = strstr(q, “ECE”); // not t
p ++;
q = p;
```

Symbol	Address	Value
q	230	A207
p	222	A207
t	214	A207
	213	\0
	212	E
	211	C
	210	E
	209	C
	208	E
	207	C
	206	E
	205	C
	204	E
	203	
t[2]	202	E
t[1]	201	C
t[0]	200	P

```
char *t = “PCE ECECECECE”;
// How many ECE does t have?
// Does “ECECE” count as one or two?
char * p;
p = strstr(t, “ECE”);
p ++;
char * q = p;
p = strstr(q, “ECE”); // not t
p ++;
q = p;
p = strstr(q, “ECE”);
```

Symbol	Address	Value
q	230	A207
p	222	A208
t	214	A200
	213	\0
	212	E
	211	C
	210	E
	209	C
	208	E
	207	C
	206	E
	205	C
	204	E
	203	
t[2]	202	E
t[1]	201	C
t[0]	200	P

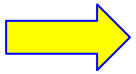
```
char *t = “PCE ECECECECE”;
// How many ECE does t have?
// Does “ECECE” count as one or two?
char * p;
p = strstr(t, “ECE”);
```

Symbol	Address	Value
p	222	A204
t	214	A200
	213	\0
	212	E
	211	C
	210	E
	209	C
	208	E
	207	C
	206	E
	205	C
	204	E
	203	
t[2]	202	E
t[1]	201	C
t[0]	200	P

```
char *t = “PCE ECECECECE”;
// How many ECE does t have?
// Does “ECECE” count as one or two?
char * p;
p = strstr(t, “ECE”);
p += strlen(“ECE”);
```

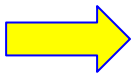
Symbol	Address	Value
p	222	A207
t	214	A200
	213	\0
	212	E
	211	C
	210	E
	209	C
	208	E
	207	C
	206	E
	205	C
	204	E
	203	
t[2]	202	E
t[1]	201	C
t[0]	200	P

```
char *t = “PCE ECECECECE”;
// How many ECE does t have?
// Does “ECECE” count as one or two?
char * p;
p = strstr(t, “ECE”);
p += strlen(“ECE”);
char * q = p;
```



Symbol	Address	Value
q	230	A207
p	222	A207
t	214	A200
	213	\0
	212	E
	211	C
	210	E
	209	C
	208	E
	207	C
	206	E
	205	C
	204	E
	203	
t[2]	202	E
t[1]	201	C
t[0]	200	P

```
char *t = “PCE ECECECECE”;
// How many ECE does t have?
// Does “ECECE” count as one or two?
char * p;
p = strstr(t, “ECE”);
p += strlen(“ECE”);
char * q = p;
p = strstr(q, “ECE”);
```



Symbol	Address	Value
q	230	A207
p	222	A208
t	214	A200
	213	\0
	212	E
	211	C
	210	E
	209	C
	208	E
	207	C
	206	E
	205	C
	204	E
	203	
t[2]	202	E
t[1]	201	C
t[0]	200	P

```
char *t = “PCE ECECECECE”;
char * p;
p = strstr(t, “ECE”);
p += strlen(“ECE”);
char * q = p;
p = strstr(q, “ECE”);
// How many ECE does t have?
// Does “ECECE” count as one or two?
// p += strlen(“ECE”) count as one
// p ++ count as two
```

Symbol	Address	Value
q	230	A207
p	222	A208
t	214	A200
	213	\0
	212	E
	211	C
	210	E
	209	C
	208	E
	207	C
	206	E
	205	C
	204	E
	203	
t[2]	202	E
t[1]	201	C
t[0]	200	P