
Data Structure: Dynamic memory allocation

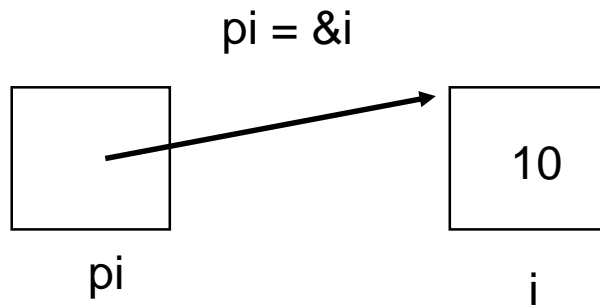
chap. 1.2, 2.1-2.3

Yongjun Park
Hanyang University



int vs. pointer-to-int

int i, *pi



i = 10

or

*pi = 10

“i” is a variable of an **integer**

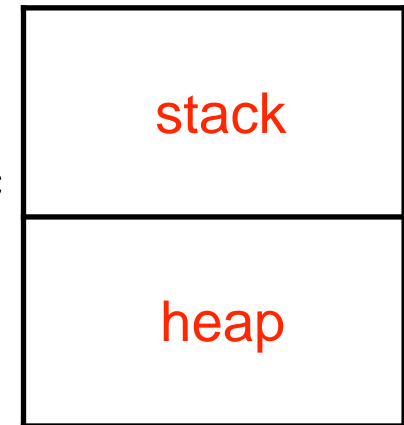
“pi” is a variable of a **pointer to an integer (address)**

“&i” returns the **address of variable i**

“*pi” returns an **integer value in the address pi**

program execution in memory

- data space consists of the stack and the heap
- the **stack** is used to store **statically declared data**
 - variables with names
 - data declared before compilation
 - access via their identifiers
- the **heap** is used to store **dynamically allocated data**
 - storage without names
 - get it when you need it
 - access by following pointers
 - by memory allocation function such as malloc



allocating and freeing dynamic data

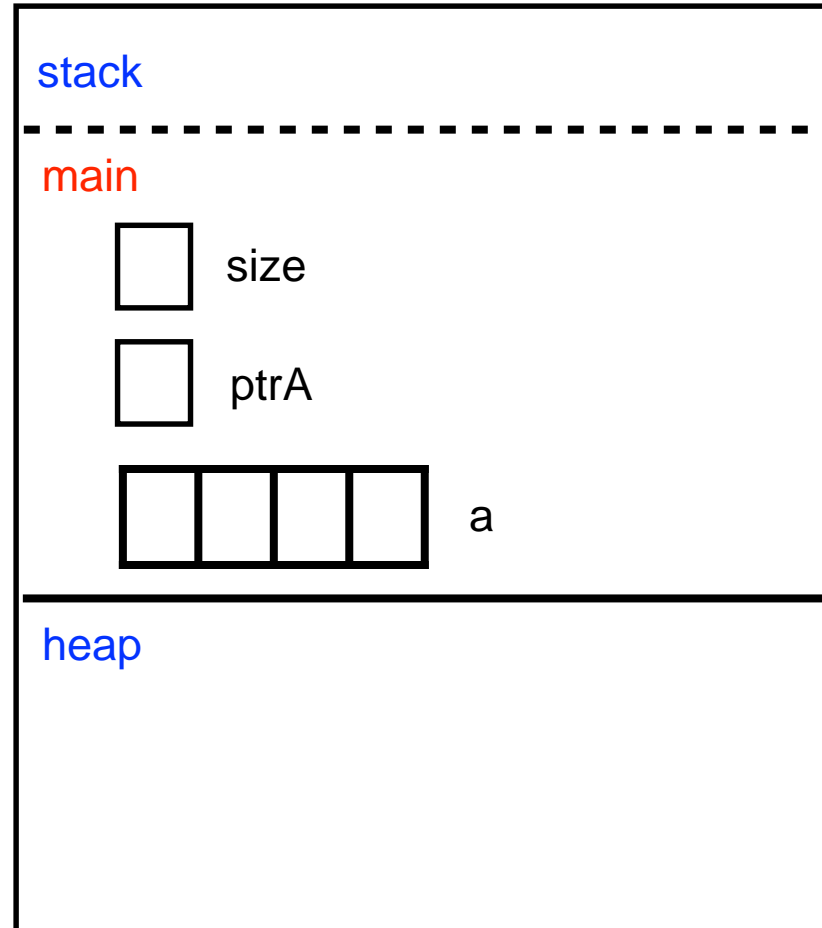
```
main()
{
    int a[4];
    int *ptrA;
    int size;
}
```

stack

heap

allocating and freeing dynamic data

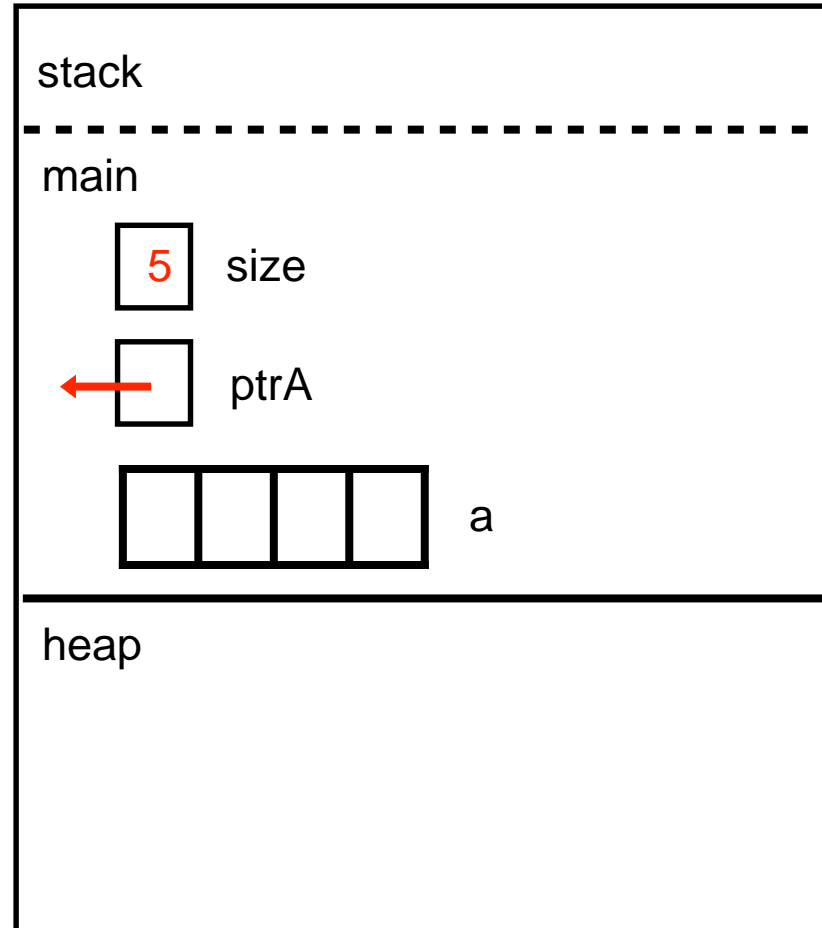
```
main()
{
    int a[4];
    int *ptrA;
    int size;
}
```



allocating and freeing dynamic data

```
main()
{
    int a[4];
    int *ptrA;
    int size;

    size = 5;
    ptrA = NULL;
}
```

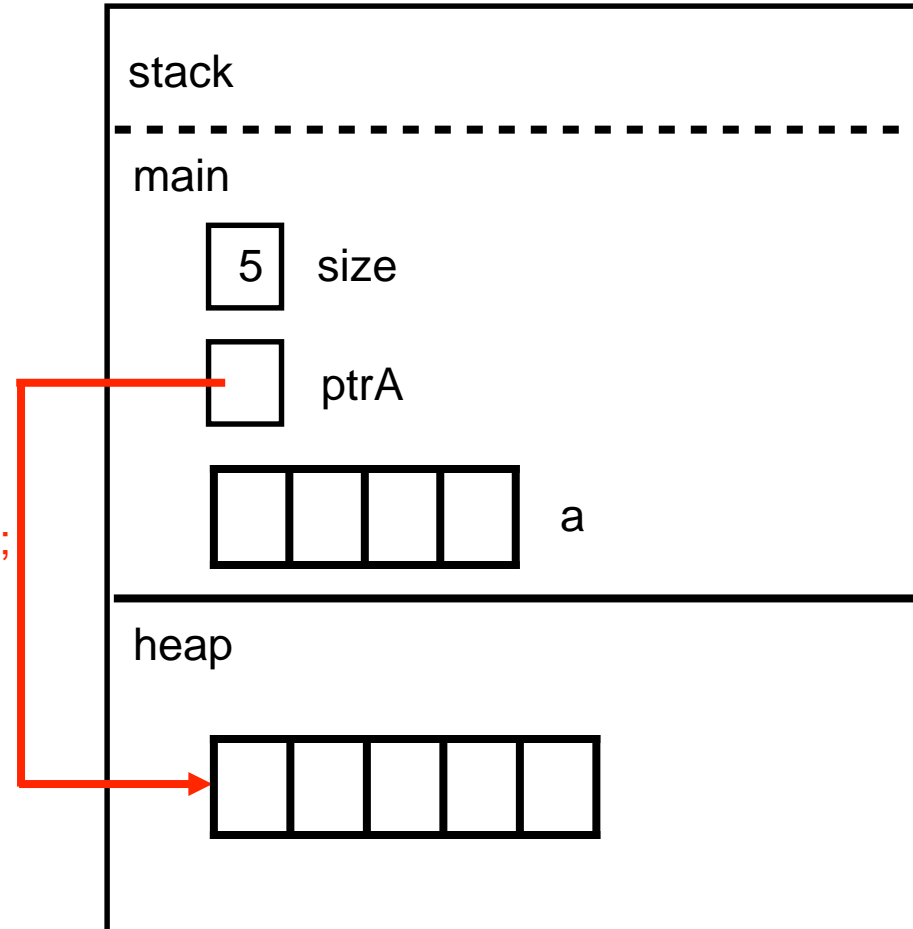


allocating and freeing dynamic data

```
main()
{
    int a[4];
    int *ptrA;
    int size;

    size = 5;
    ptrA = NULL;

    ptrA = (int *) malloc(size * sizeof(int));
}
```



allocating and freeing dynamic data

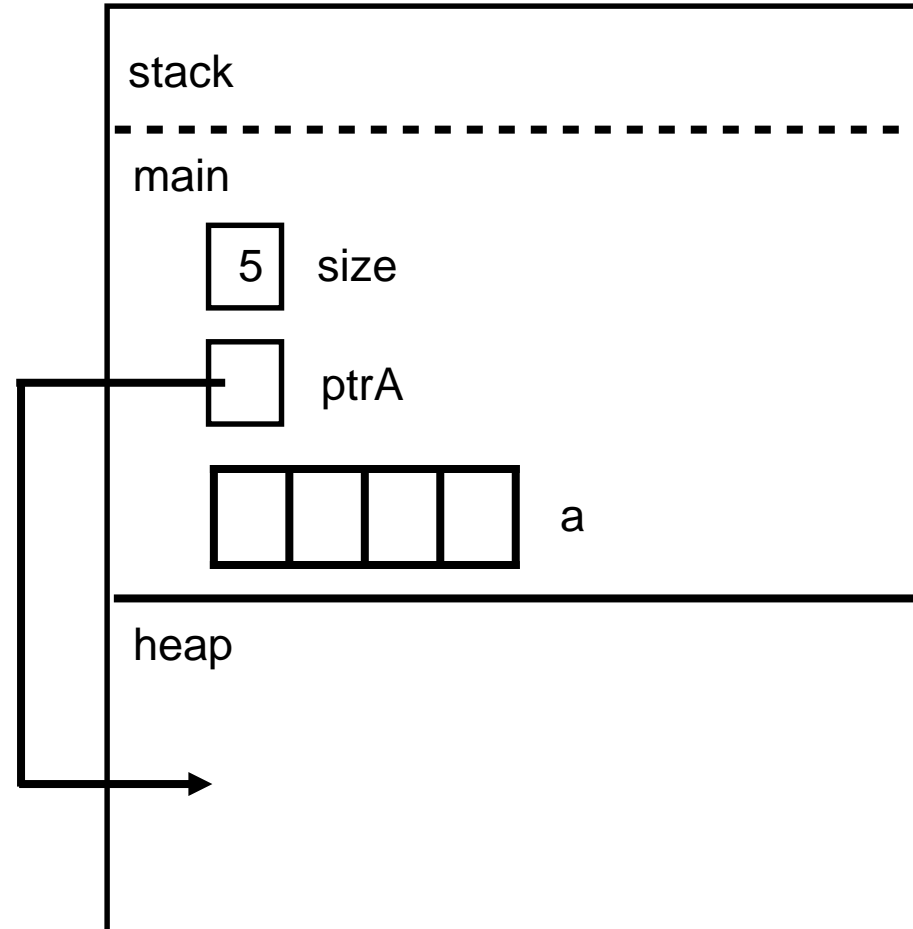
```
main()
{
    int a[4];
    int *ptrA;
    int size;

    size = 5;
    ptrA = NULL;

    ptrA = (int *) malloc(size * sizeof(int));

    free(ptrA);

}
```



allocating and freeing dynamic data

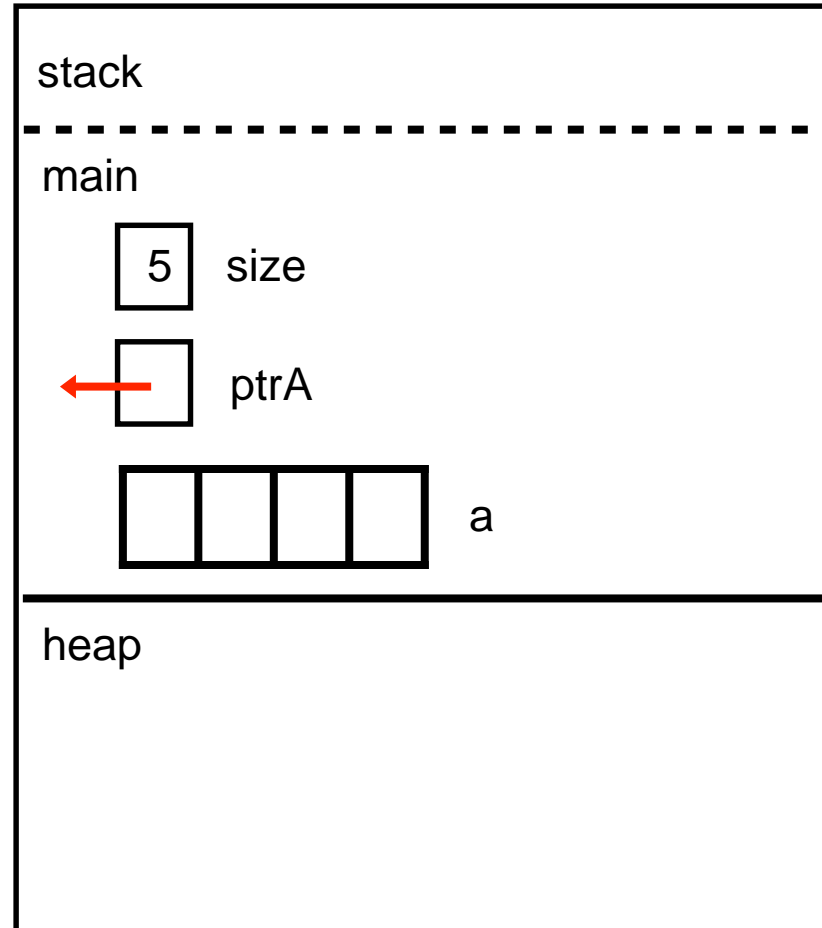
```
main()
{
    int a[4];
    int *ptrA;
    int size;

    size = 5;
    ptrA = NULL;

    ptrA = (int *) malloc(size * sizeof(int));

    free(ptrA);

    ptrA = NULL;
}
```



dynamic allocation

*void *malloc (size_t size)*

- it returns a pointer to space for **an object of size size** or NULL if the request cannot be satisfied
- `intPtr = (int *) malloc (size * sizeof(int))`

*void realloc(void *p, size_t size)*

- it **changes the size of the object** pointed to by *p* to *size*
- the contents will be unchanged up to the minimum of the old and new sizes
- `intPtr = (int *) realloc(intPtr, 50)`

*void free (void *p)*

- it **deallocates the space pointed to by p**
- *p* must be a pointer to space previously allocated by *malloc*, or *realloc*

array

- Is it OK?

```
#include <stdio.h>

void main(void){

    int *list1;
    int list2[5];

    list1[0] = 34;
    list2[0] = 34;

}
```



array

- Is it OK?

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

void main(void){

    int *list1;
    int list2[5];

    list2[0] = 34;

    list1 = (int *)malloc(5*sizeof(int));

    list1[0] = 35;
    printf("%d  %d\n", list1[0], list2[0]);

}
```

structures

- Example: storing information about persons including
 - Name
 - Age
 - Height

structures

- Example: storing information about persons including
 - Name
 - Age
 - Height
- a **structure** is a collection of one or more variables that can be of different types
- How?
 - First, create a structure that defines a new data type
 - Second, create **variable** of that new type

```
typedef struct {  
    char *name;  
    int age;  
    double height;  
} personT;
```



structures

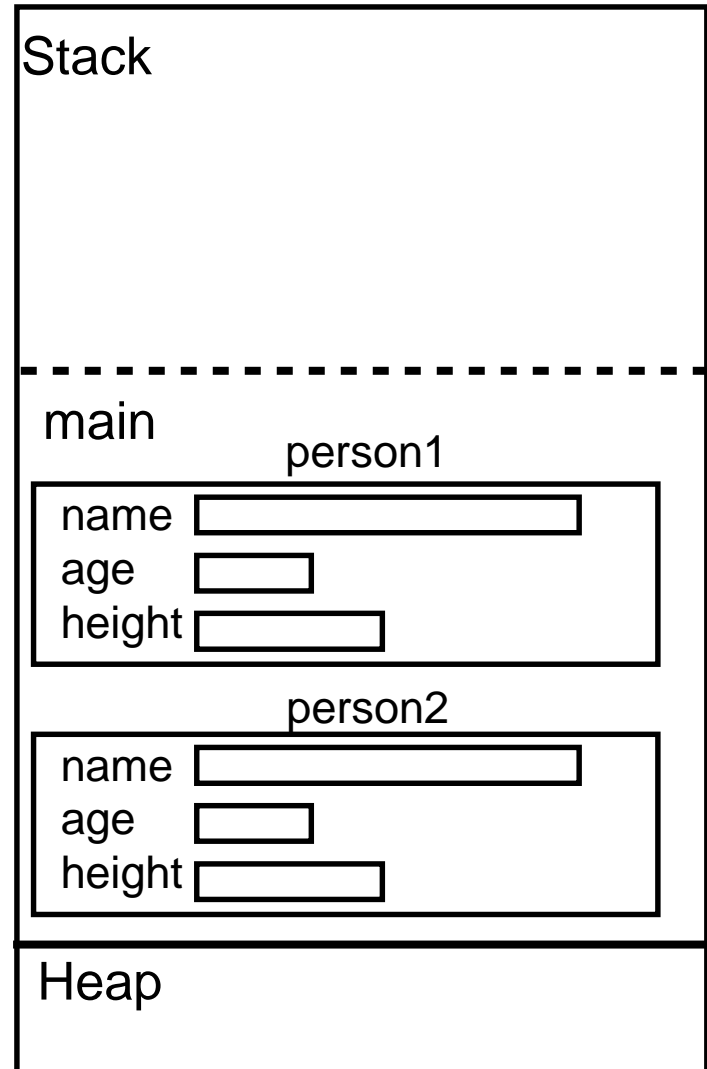
```
typedef struct {  
    char *name;  
    int age;  
    double height;  
} personT;
```

```
main()  
{  
    personT person1;  
  
    person1.name = "Brian";  
    person1.age = 10;  
    person1.height = 20;  
}
```



structures: passing structure

```
typedef struct {  
    char *name;  
    int age;  
    double height;  
} personT;  
void GetPersonData(personT x);  
  
main()  
{  
    personT person1;  
    personT person2;  
  
    GetPersonData(person1);  
}  
  
void GetPersonData (personT x){  
    x.name = "Brian";  
    x.age = 10;  
    x.height = 20;  
}
```



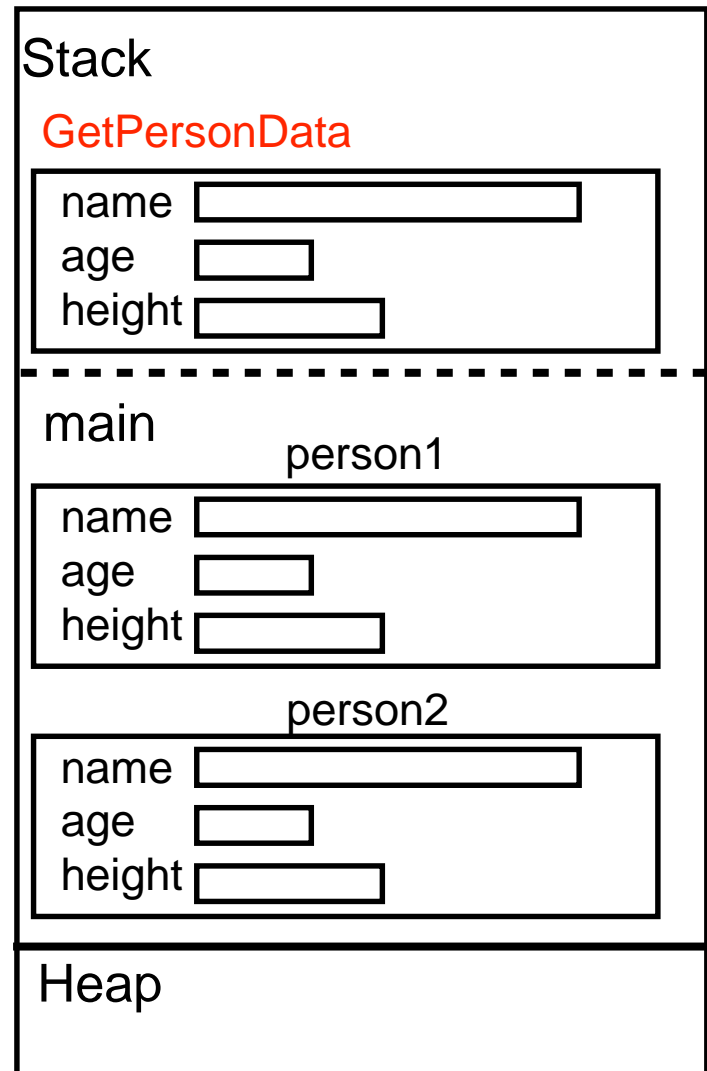
structures: passing structure

```
typedef struct {
    char *name;
    int age;
    double height;
} personT;
void GetPersonData(personT x);

main()
{
    personT person1;
    personT person2;

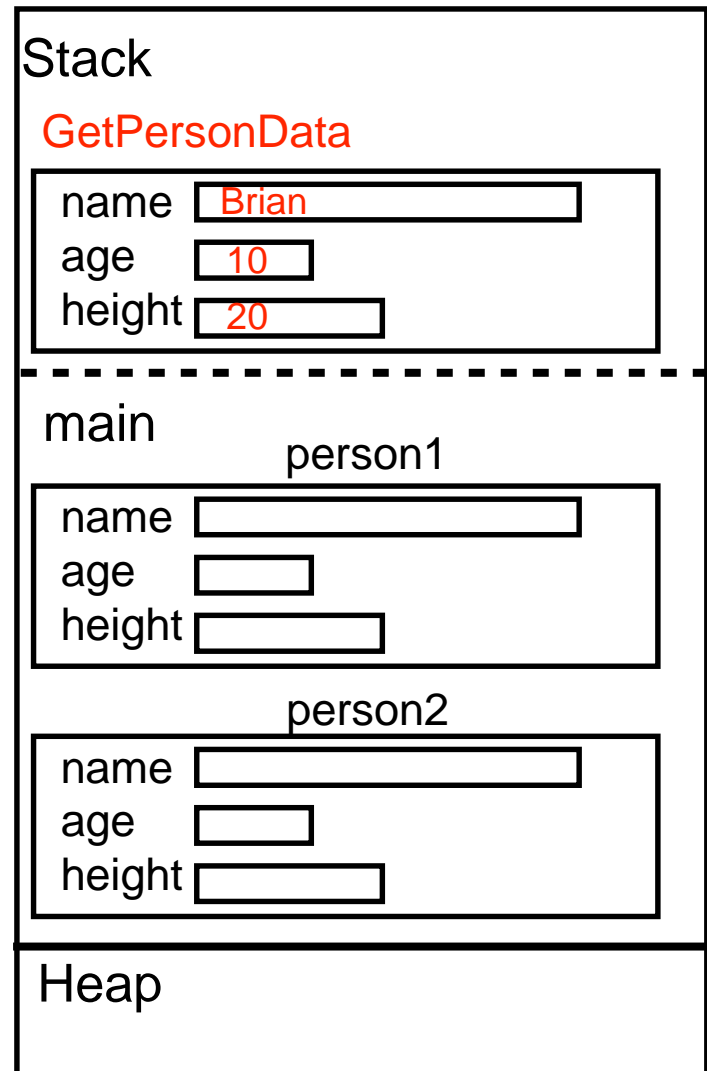
    GetPersonData(person1);
}

void GetPersonData (personT x){
    x.name = "Brian";
    x.age = 10;
    x.height = 20;
}
```



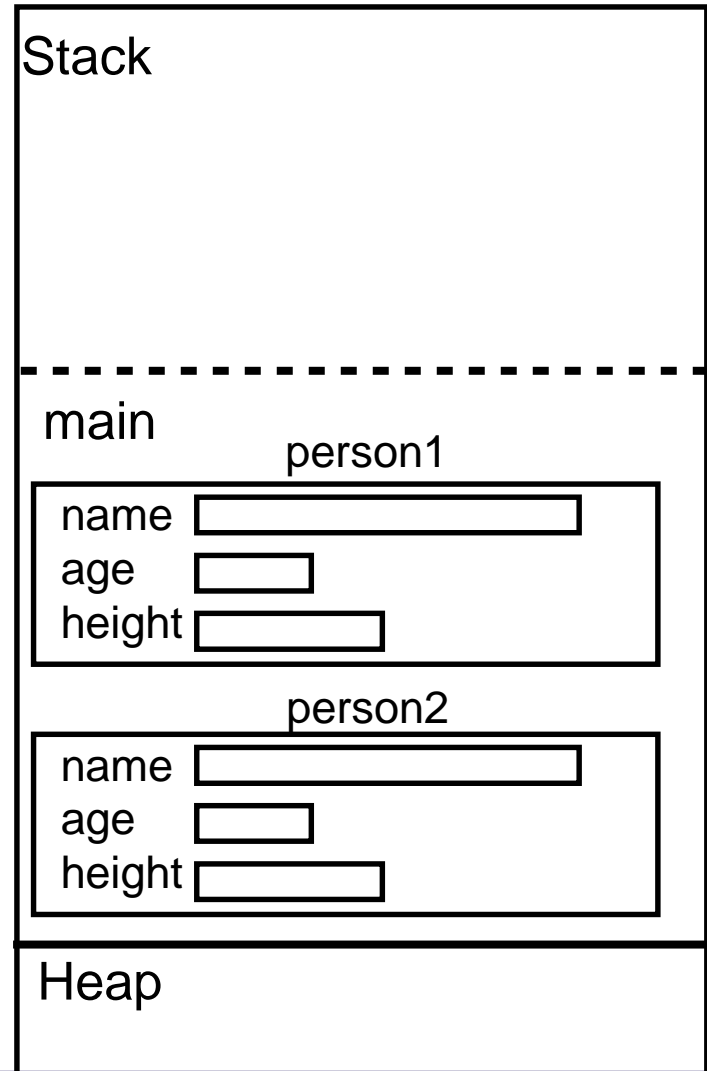
structures: passing structure

```
typedef struct {  
    char *name;  
    int age;  
    double height;  
} personT;  
void GetPersonData(personT x);  
  
main()  
{  
    personT person1;  
    personT person2;  
  
    GetPersonData(person1);  
}  
  
void GetPersonData (personT x){  
    x.name = "Brian";  
    x.age = 10;  
    x.height = 20;  
}
```



structures: passing structure

```
typedef struct {  
    char *name;  
    int age;  
    double height;  
} personT;  
void GetPersonData(personT x);  
  
main()  
{  
    personT person1;  
    personT person2;  
  
    GetPersonData(person1);  
}  
  
void GetPersonData (personT x){  
    x.name = "Brian";  
    x.age = 10;  
    x.height = 20;  
}
```



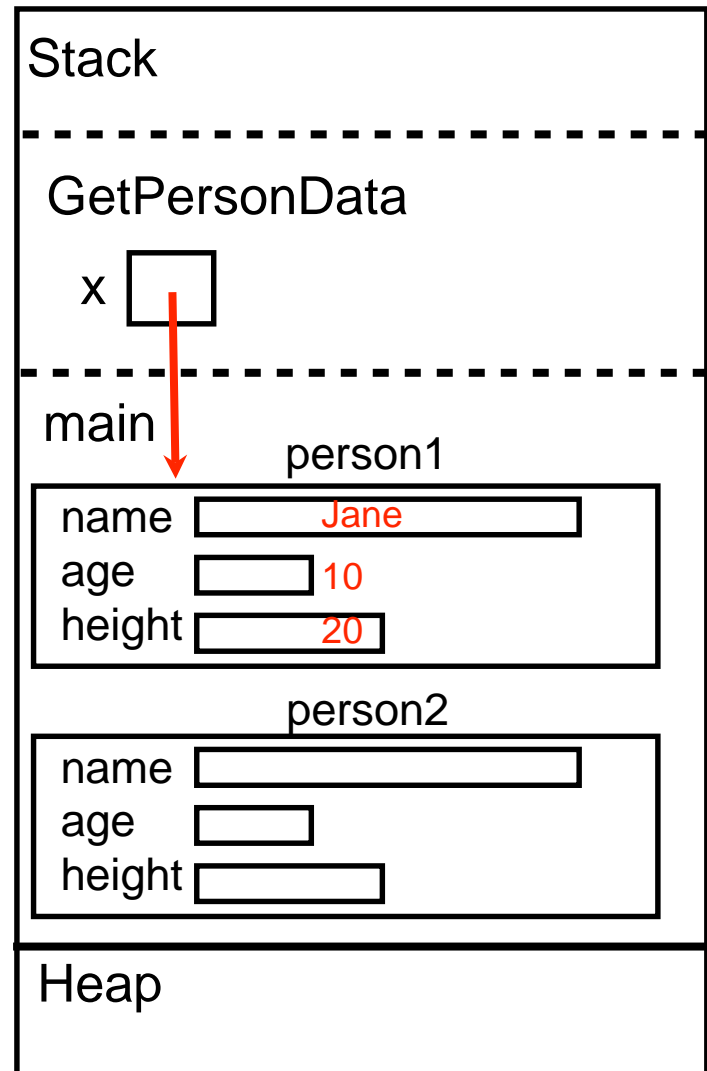
structures: passing address

```
typedef struct {
    char *name;
    int age;
    double height;
} personT;
void GetPersonData(personT *x);

main()
{
    personT person1;
    personT person2;

    GetPersonData(&person1);
}

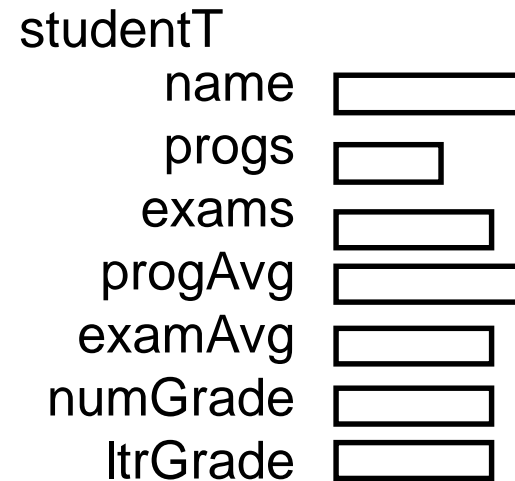
void GetPersonData (personT *x){
    x->name = "Jane";
    x->age = 10;
    x->height = 20;
}
```



creating data structure with structure

```
#define NUM_HW 6
#define NUM_EXAMS 2

typedef struct {
    string name;
    int progs [NUM_HW];
    int exams [NUM_EXAMS];
};
int progAvg;
double examAvg;
double numGrade;
string ltrGrade;
} studentT;
```



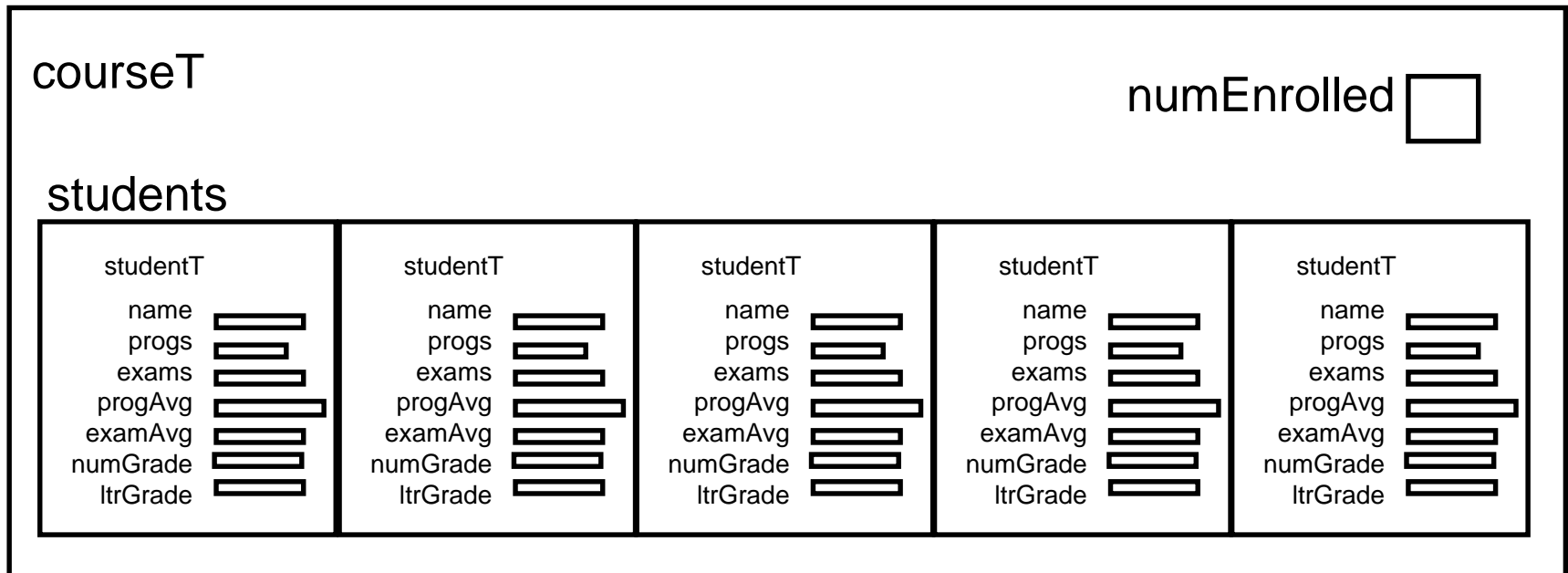
A diagram illustrating the memory layout of the `studentT` structure. It shows the following fields and their relative sizes represented by horizontal bars:

- `name`: A long bar representing a string.
- `progs`: A short bar representing an array of 6 integers.
- `exams`: A short bar representing an array of 2 integers.
- `progAvg`: A long bar representing an integer.
- `examAvg`: A short bar representing a double.
- `numGrade`: A short bar representing a double.
- `ltrGrade`: A short bar representing a string.

creating data structure with structure

```
#define MAX_ENROLL 5
```

```
typedef struct{  
    studentT students[MAX_ENROLL];  
    int numEnrolled;  
} courseT;
```



creating data structure with structure

```
main()
{
    courseT cs106A;    /* allocates memory on stack */
    int i;

    cs106A.numEnrolled = 0;

    for (i=0; i<MAX_ENROLL; i++){
        cs106A.students[i] = GetStudentData();
        cs106A.numEnrolled++;
    }
}
```

creating data structure with structure

```
main()
{
    courseT *cs106A;
    int i;

    cs106A = (courseT *)malloc(sizeof(courseT));    /* allocates i
n heap */
    cs106A -> numEnrolled = 0;

    for (i=0; i<MAX_ENROLL; i++){
        cs106A -> students[i] = GetStudentData();
        cs106A -> numEnrolled++;
    }
}
```