

CS601: Software Development for Scientific Computing

Autumn 2023

Week3: Programming Environment (contd),
Makefile

C++ standard types

- Integer types: `char`, `short int`, `int`, `long int`, `long long int`, `bool`
- Float: `float`, `double`, `long double`
- Pointers: handle to addresses
- References: safer than pointers but less powerful
- `void`: nothing

C++ standard types

- Compound types
 - pointers, structs, enums, arrays, etc.
- Modifiers
 - short, long, signed, unsigned.

types / representation

E.g. `int x;`

1. What is the set of values this variable can take on in C?

-2^{31} to $(2^{31} - 1)$

2. How should operations on this variable be handled?
integer division is different from floating point divisions

`3 / 2 = 1 //integer division`

`3.0 / 2.0 = 1.5 //floating-point division`

3. How much space does this variable take up?

32 bits

C++ standard types – storage space

Data type	Number of bytes
char	1
short int	2
int / long int	4
long long int	8
float	4
double	8
long double	12

- All built-in types are represented in memory as a contiguous set of bytes
- Use sizeof() operator to check the size of a type
 - e.g. sizeof(int)

Typedef

- Lets you give alternative names to C data types
- Example:

```
typedef unsigned char BYTE;
```

This gives the name BYTE to an unsigned char type.
Now,

```
BYTE a;
```

```
BYTE b;
```

Are valid statements.

Typedef Syntax

```
typedef [existing_type] [new_type];
```

- Resembles a definition/declaration without initializer;

E.g.

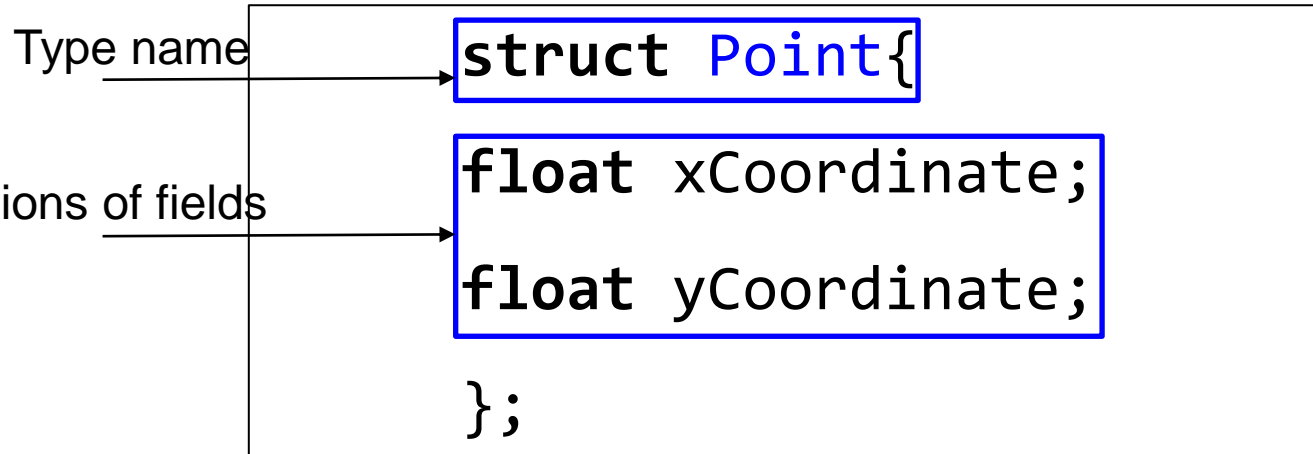
```
int [x];
```

- Mostly used with user-defined types

User-defined Types

- *Structures* in C/C++ are one way of defining your own type.
- Arrays are compound types but have the *same* type within.
 - E.g. A string is an array of char
 - `int arr[]={1,2,3};` arr is an array of integer types
- Structures let you compose types with *different* basic types within.

Structures - Declaration

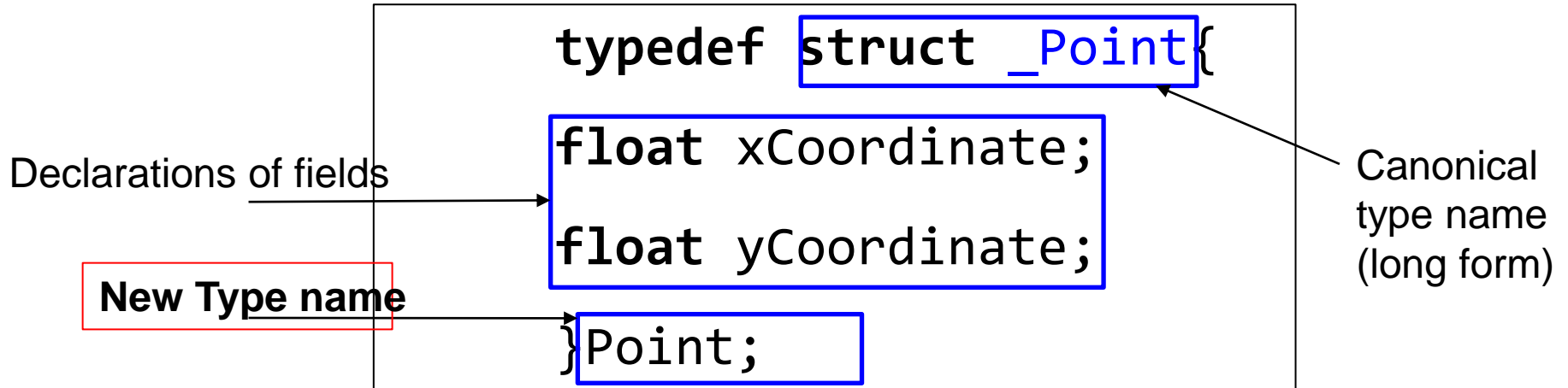


– Variable definition:

- `struct Point p1;`
- `struct Point{
 float xCoordinate;
 float yCoordinate;
}p1;`

`p1` is a variable (an object) of type `struct Point`,

Structures - Definition



- Variable definition:
 - `Point p1;`

Structures - Usage

- Structure fields are accessed using dot (.) operator
- Example:

```
Point p;
```

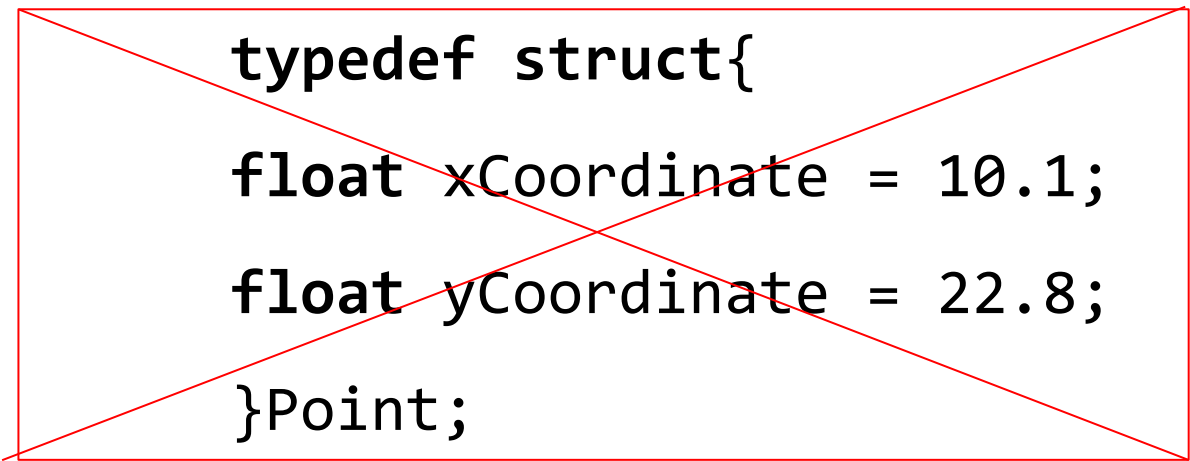
```
p.xCoordinate = 10.1;
```

```
p.yCoordinate = 22.8;
```

```
printf("(x,y)=(%f,%f)\n",p.xCoordinate,  
p.yCoordinate);
```

Structures - Initialization

- Error to initialize fields in declaration;



```
typedef struct{  
    float xCoordinate = 10.1;  
    float yCoordinate = 22.8;  
}Point;
```

Data types - quirks

- if no type is given compiler automatically converts it to `int` data type.
 - `signed x;`
- `long` is the only modifier allowed with `double`
 - `long double y;`
- `signed` is the default modifier for `char` and `int`
- Can't use any modifiers with `float`

Exercise

```
char s[3] = "Hi";
```

```
char *t = "Si";
```

```
int u[3] = {5, 6, 7};
```

```
int n=8;
```

Expression	Type	Comments
s	char[3]	array of 3 chars
t	char*	address of a char
u	int[3]	array of 3 ints
&u[0]	int*	address of an int

Exercise

```
char s[3] = "Hi";
```

```
char *t = "Si";
```

```
int u[3] = {5, 6, 7};
```

```
int n=8;
```

Expression	Type	Comments
*&n	int	value at n
*t	char	data at address Held by t

Exercise

- Array initializers:

1. `int u[3] = {5, 6};`

Is this valid?

If yes, what is the value held in the third element `u[2]`?

2. `int u[3] = {5, 6, 7, 8};`

Is this valid?

3. `char s1[]="Hi";`

What is the size of `s1`? (how many bytes are reserved for `s1`)

4. `char s2[3]="Si";`

Is this valid?

Exercise

```
int u[3] = {5, 6, 7};  
int* p=u;  
p[0]=7;  
p[1]=6;  
p[2]=5;
```

//Now, u would contain the numbers in reverse order.
u[0] = 7, u[1]=6, u[2]=5.

```
char *str = "Hello";  
char* p=str;  
p[0]='Y';  
//Now, what would str contain?
```

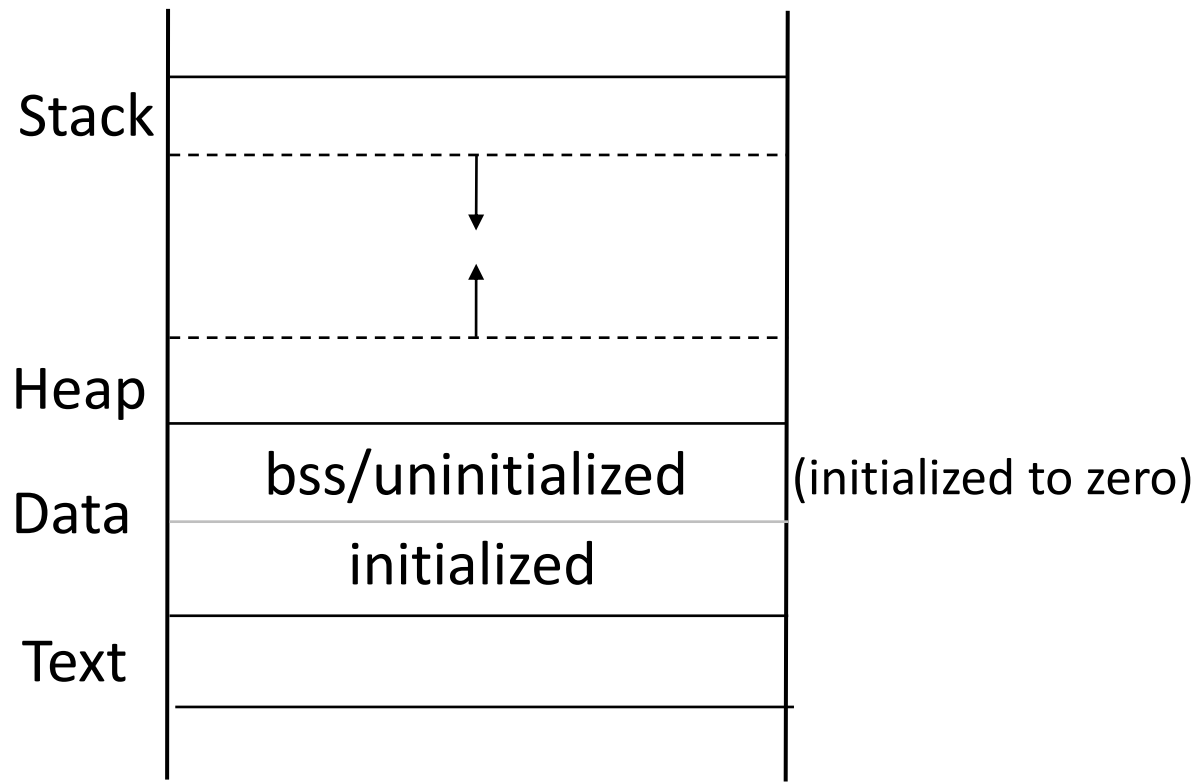
Program layout in memory

- How is a program laid out in memory?
 - Helpful to debug
 - Helpful to create robust software
 - Helpful to customize program for embedded systems

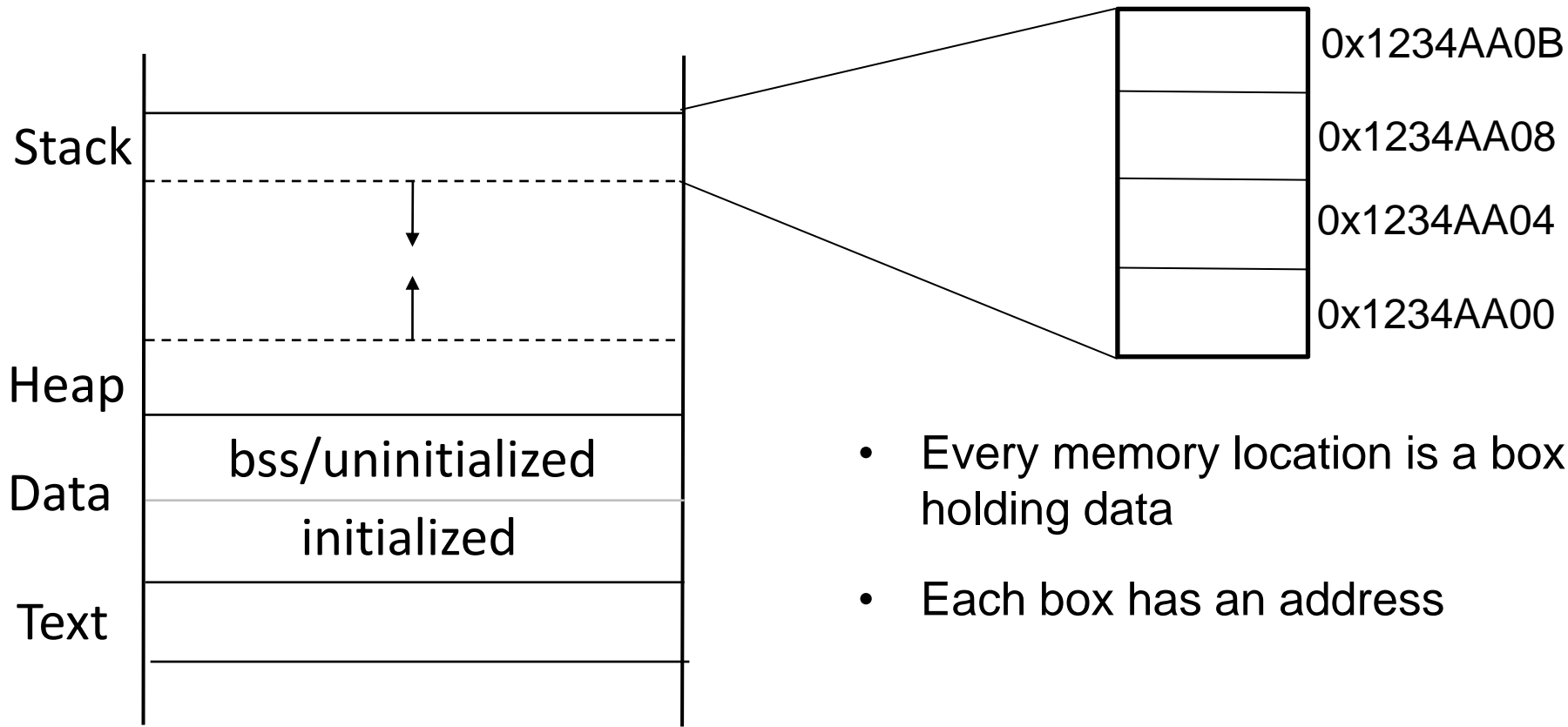
Program Layout in Memory

- A program's memory space is divided into four segments:
 1. Text
 - source code of the program
 2. Data
 - Broken into uninitialized and initialized segments; contains space for global and static variables. E.g. `int x = 7; int y;`
 3. Heap
 - Memory allocated using `malloc/calloc/realloc/new`
 4. Stack
 - Function arguments, return values, local variables, [special registers](#).

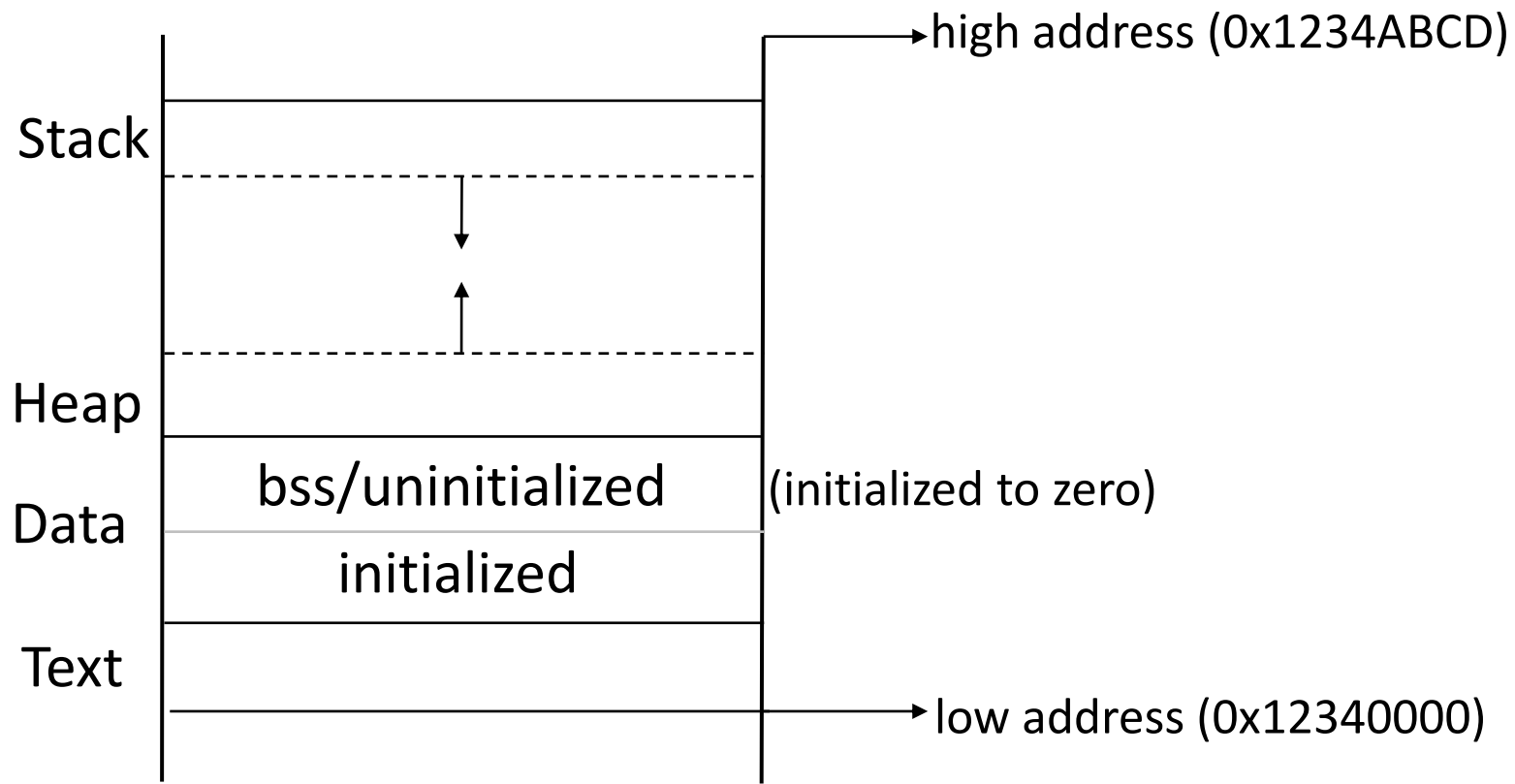
Program Layout in Memory



Program Layout in Memory



Program Layout in Memory



Exercise

- Write a C++ program with the following requirements:
 - User should be able to provide the dimension of two vectors (*do not use C++ vectors from STL*)
 - The program should allocate two vectors of the required size and initialize them with meaningful data
 - The program should compute the scalar product of the two vectors and print the result

Discussion

Refer to:

- `vectorprod_v1.cpp`
 - What if `atoi` doesn't provide accurate status about the value returned?
- `vectorprod_v2.cpp`
 - C++ `stringstreams` are an option. Is this code modular?
- `vectorprod_v3.cpp` `scprod.cpp`
 - What if there is already built-in function by the same name?
- `vectorprod_v4.cpp` `scprod_v4.cpp`
 - Namespaces